Network-Agnostic Security Comes for Free in DKG and MPC

Renas Bacho[1*], Daniel Collins[2†], Chen-Da Liu-Zhang[3‡], and Julian Loss[1]

[1] CISPA Helmholtz Center for Information Security
{renas.bacho,loss}@cispa.de
[2] EPFL
daniel.collins@epfl.ch
[3] NTT Research
chen-da.liuzhang@ntt-research.com

Abstract. Distributed key generation (DKG) protocols are an essential building block for threshold cryptosystems. Many DKG protocols tolerate up to $t_s < n/2$ corruptions assuming a well-behaved synchronous network, but become insecure as soon as the network delay becomes unstable. On the other hand, solutions in the asynchronous model operate under arbitrary network conditions, but only tolerate $t_a < n/3$ corruptions, even when the network is well-behaved.
In this work, we ask whether one can design a protocol that achieves security guarantees in either scenario. We show a complete characterization of *network-agnostic* DKG protocols, showing that the tight bound is $t_a + 2t_s < n$. As a second contribution, we provide an optimized version of the network-agnostic MPC protocol by Blum, Liu-Zhang and Loss [CRYPTO'20] which improves over the communication complexity of their protocol by a linear factor. Moreover, using our DKG protocol, we can instantiate our MPC protocol in the *plain PKI model*, i.e., without the need to assume an expensive trusted setup.
Our protocols incur the same communication complexity as state-of-the-art DKG and MPC protocols with optimal resilience in their respective purely synchronous and asynchronous settings, thereby showing that network-agnostic security comes *for free*.

1 Introduction

The problem of *distributed key generation* (DKG) has been extensively studied in the cryptographic literature and is a fundamental building block for threshold cryptosystems. It allows a set of n parties to compute a uniform sharing

[*]The author was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 507237585.

[†]This work was partially carried out while the author was visiting CISPA.

[‡]This work was partially carried out while the author was at CMU and supported by the NSF award 1916939, DARPA SIEVE program, a gift from Ripple, a DoE NETL award, a JP Morgan Faculty Fellowship, a PNC center for financial services innovation award, and a Cylab seed funding award.

of a secret key such that a sufficiently large threshold of $t < n$ parties must cooperate to reconstruct the secret or compute some function of it. As such, DKG has met many applications, including key escrow services, password-based authentication, threshold signing and encrypting, and many more.

Many existing protocols solve DKG for up to $t < n/2$ malicious corruptions, assuming that the network is *synchronous* [Ped91,GJKR99,SBKN21]. In the synchronous model, message delays are upper bounded by some known delay Δ and parties are assumed to have synchronized clocks. These protocols, however, provide no security guarantees when the network is *asynchronous*. Therefore, a more recent line of work has aimed at solving DKG in the asynchronous network model [KG09,AJM$^+$21,DYX$^+$22]. However, asynchronous protocols inherently tolerate at most $t < \frac{n}{3}$ malicious parties, even when the network is synchronous. This poses a vexing dilemma for a protocol designer who can not predict the behaviour of the network. On the one hand, she can choose a synchronous protocol that tolerates the maximum number of $t < n/2$ malicious parties. However, such a protocol might lose all security guarantees if the network ever becomes asynchronous. On the other hand, she can opt for an asynchronous protocol. While this type of protocol remains secure under arbitrary network conditions, it tolerates only $t < n/3$ corrupted parties *even if the network behaves synchronously.*

In this work, we ask the following question: Is it possible to design a *network-agnostic* DKG protocol that achieves security guarantees in either scenario? Moreover, can we achieve network-agnostic security with no efficiency overhead, i.e. with the same efficiency as state-of-the-art purely synchronous/asynchonous DKG schemes?

We answer these questions in the affirmative. Our contributions are motivated by a series of recent works on network-agnostic protocols for various types of consensus and MPC. Existing protocols, however, strongly rely on trusted setup, particularly in the form of threshold cryptosystems. Thus, the import of our work lies within replacing this setup at essentially no cost. In more detail, we show the following results:

- We propose the first network-agnostic DKG protocol. Our protocol tolerates $n/3 < t_s < n/2$ corrupted parties in the synchronous model and $t_a < n/3$ parties in the asynchronous model where t_a and t_s can be chosen arbitrarily subject to $t_a + 2 \cdot t_s < n$. Our protocol works in the plain PKI model[4] and allows parties to agree on a field element x for a $y = g^x$ with $O(\lambda n^3)$ communication complexity. This matches the best-known results in synchrony [SBKN21] and asynchrony [DYX$^+$22]. Thus, our DKG protocol can be used to efficiently bootstrap trusted key generation for network-agnostic consensus and MPC protocols.
- As an application of our DKG protocol, we obtain the first network-agnostic MPC protocol with optimal resilience $t_a + 2 \cdot t_s < n$ in the plain PKI model. Moreover, as a second contribution we show an optimized version of network-agnostic MPC with communication complexity of $O(|C|n^2)$ field elements to

[4]In this model, the public keys from corrupted parties can be generated arbitrarily.

evaluate a circuit C, improving a linear factor over the state of the art. This matches the most efficient purely asynchronous MPC protocol (in the setting without the use of multiplicative-homomorphic encryption, see the related work for details).

In summary, our protocols incur no additional setup assumptions or asymptotic overhead over state-of-the-art protocols for the synchronous and asynchronous network models. This shows that network-agnostic DKG and MPC essentially come 'for free'.

1.1 Background and Starting Point

We consider n parties $P_1, \ldots, P_n$ that communicate over pairwise authenticated channels. Moreover, we assume that parties share a public key infrastructure (PKI), and denote P_i's secret and public key, respectively, as $\mathsf{sk}_i, \mathsf{pk}_i$. We do not make any assumption on the distributions of corrupted parties' keys and assume they can be maliciously generated. Throughout, we fix thresholds $0 < t_a < \frac{n}{3} \leq t_s < \frac{n}{2}$ such that $t_a + 2 \cdot t_s < n$. Our model assumptions can now be characterized as follows:

– If the model is synchronous, parties are assumed to have synchronized clocks and messages sent by parties are delivered within some known upper bound Δ. At most t_s parties can be maliciously corrupted.
– Otherwise, if the network is asynchronous, messages can be arbitrarily delayed, as long as they are never dropped and are delivered within finite time. Moreover, parties' clocks can be arbitrarily out of sync. In this case, at most t_a parties may be maliciously corrupted. Note that the network can never become asynchronous once t_a or more parties have been corrupted.
– Parties do not know a priori how the network might behave.

Our goal is to design a distributed key generation (DKG) protocol for parties to securely distribute a uniform *field* element x corresponding to some $y = g^x$. Since parties cannot be sure whether the network is synchronous or not in general, we require that the secret reconstruction threshold is $\ell = t_s + 1$. With the aim of matching the best known DKG protocols for the synchronous and asynchronous model, we aim for our DKG to run in $O(\lambda n^3)$ communication complexity and be statically secure. In Table 1, we compare the existing state-of-the-art with our proposed DKG which, as we show, satisfies these aims.

We stress that this goal cannot be achieved by simply running a generic, network-agnostic MPC protocol [BLL20,DHLZ21], since these protocols require trusted setup in the form of shared keys (which is exactly the goal we are trying to achieve).

Network-Agnostic Protocols: A Blueprint. To design an efficient network-agnostic DKG protocol, a natural approach is to follow the template of previous works [BKL19,BLL20,BKL21]. Here, the protocol is divided into two components, a synchronous component Π_s and an asynchronous component Π_a. Parties begin by running Π_s which performs securely, given that up to t_s parties are

DKG Protocol	**Network**	**CC**	**RC**	**Setup**
Shrestha et al. [SBKN21]	sync	$O(\lambda n^3)$	$O(n)/O(1)$	RO
Abraham et al. [AJM$^+$21]	async	$O(\lambda n^3)$	$O(1)$	PKI, CRS
Das et al. [DYX$^+$22]	async	$O(\lambda n^3)$	$O(\log n)$	PKI, RO
Section 3	fallback	$O(\lambda n^3)$	$O(n)$	PKI, CRS/RO

Table 1: Performance comparison of state-of-the-art DKG protocols. The network is synchronous (sync), asynchronous (async) or either synchronous or asynchronous (fallback). We measure complexity via communication (CC) and round (RC) complexity. Setup assumptions include a bulletin board PKI (PKI), a random oracle (RO) or a common reference string (CRS). Abraham et al. [AJM$^+$21]'s DKG constructs a shared *group* element, rather than a *field* element.

corrupted. In this case, parties pass the output v_s obtained from Π_s to Π_a. The final output of the protocol is the value v_a output by Π_a. Note, however, that Π_a achieves security only against t_a corruptions, as it is asynchronous. Thus, the key challenge is to prevent Π_a from simply overwriting the output v_s in a synchronous network, as this would degrade the overall corruption threshold of the protocol to t_a.

To prevent this outcome, the idea is to design Π_s and Π_a with two special properties. First, suppose that the network is synchronous and parties agree on the intermediate value v_s passed to the asynchronous component Π_a. In this case, Π_a should simply relay the correct output v_s (rather than recomputing it), *even if t_s parties are corrupted.* Second, if the network is asynchronous, Π_a should be able to compute the correct output v_a on its own in the presence of t_a corruptions. In addition to this, Π_s must prevent parties from computing a catastrophically incorrect intermediate output v_s that might violate the overall security properties of the protocol, given an asynchronous network with t_a corrupted parties.

Background: Synchronous DKG. The above discussion shows why naively running a synchronous DKG and then an asynchronous agreement protocol back-to-back would not produce a network-agnostic protocol. For the setting of DKG, this might lead to the resulting secret key not having enough 'contributions' from honest parties. Our starting point is the synchronous New-DKG protocol of Gennaro et al. [GJKR07], which we make amenable to our network model. Loosely speaking, New-DKG is divided into two phases as follows:

- **Sharing Phase.** In the first phase, each party performs verifiable secret sharing (VSS) using Pedersen's VSS scheme [Ped92] to share two random polynomials f and f'. Parties then execute a public complaint management protocol, since some parties may try to misbehave and, e.g., not send (correct) shares to some parties. They agree on a set of parties Q that honestly executed Pedersen's VSS.
- **Reconstruction Phase.** In the second phase, parties then reconstruct their share of the final secret. To do so, they perform the reconstruction phase of

Feldman's VSS [Fel87] from the sharing phase with respect to polynomial f. The shares of misbehaving parties are reconstructed publicly to ensure termination.

By committing to a second polynomial f', Pedersen's VSS ensures that shared secrets are unconditionally hiding and parties can efficiently blindly evaluate f in the exponent to verify their shares. Then, Q is sufficiently large to ensure that at least one party must have honestly performed VSS. Thus, the adversary has no information about the secret when Q is decided.

One may be tempted to design a simpler and more efficient DKG protocol where parties, as in the so-called Joint-Feldman DKG [GJKR07], run Feldman's VSS in parallel. Gennaro et al. [GJKR07] highlighted that the adversary can bias the distribution of the public key by manipulating the set Q, thus precluding security. In general, a rushing adversary can choose to include or exclude the contributions of parties in the final secret which thus precludes any 'one-round' DKG protocol from outputting a uniformly random secret.[5]

In any case, it is not hard to see that the complaint management protocols in New-DKG fail in asynchrony. This is because the complaints of honest parties may be arbitrarily delayed on the network, precluding either correctness or liveness. Dealing with this issue creates additional challenges which we address in the following section.

1.2 Technical Overview: DKG

A natural idea is to replace Pedersen's VSS in the first phase with publicly verifiable secret sharing (PVSS) [CGMA85,Sta96]. The key property of PVSS is that all parties can *non-interactively* verify whether a given sharing is correct. We begin by presenting a secure, but excessively expensive strawman solution which will serve as our starting point.

A Strawman Solution. To ensure that parties agree on PVSS sharings, each party (acting as the dealer) would first synchronously broadcast their PVSS sharing. In the second phase, parties could then use the asynchronous common subset (ACS) protocol of Blum et al. [BKL21] to agree on a common subset of such sharings. In their protocol, each party provides an input v and the protocol lets them agree on a common subset of $n - t_a$ outputs, given t_a corruptions in an asynchronous network. In addition, their protocol guarantees that if all honest parties start with the *same* input v, then the protocol will remain secure for up to t_s corruptions and the output will be the singleton set $\{v\}$. These properties have made their protocol a staple building block in many network-agnostic protocols.

In our scenario, parties would input their common view of all sharings after the broadcast phase to ACS. Given that the synchronous phase succeeded (i.e., the network is synchronous), all parties would input the *same view* and hence ACS would allow them to (re-)agree on this view, given at most t_s corrupted

[5]Note that a possibly biased secret can still be sufficient for applications like threshold signatures, as highlighted in [GJKR07,BL22].

parties. If parties have not obtained any output from the synchronous phase, they would simply input their PVSS dealing as an input to ACS directly. Even in case the network is asynchronous, parties would still be able to agree on a subset of $n - t_a$ dealings in this manner. From this, they could securely derive a common secret key[6].

Unfortunately, existing network-agnostic ACS protocols [BKL21,ABKL22] execute n instances of binary consensus and consequently require $O(n)$ distributed coin flips. As the best known protocol to flip coins without trusted setup requires $O(\lambda n^3)$ communication complexity [GLL+21], this step alone incurs at least $O(\lambda n^4)$ overhead. In addition, the above solution requires all parties to broadcast $O(\lambda n)$-sized sets of ciphertexts containing the parties PVSS dealings. Using existing broadcast protocols, this step would incur an additional communication overhead of $O(\lambda^2 n^4)$. Towards building a network-agnostic DKG with $O(\lambda n^3)$ communication complexity, we introduce novel techniques to overcome the above challenges.

From ACS to Intrusion-Tolerant Consensus. Recall that under synchrony, all parties are guaranteed to output at least $n - t_s$ values from the parallel broadcast in our above strawman protocol. However, if the network is asynchronous, parties are not guaranteed agreement or termination of sufficiently many broadcast instances. To cope, our idea is to let parties execute an asynchronous agreement protocol with an *intrusion tolerance* validity property [MR10]. Intrusion tolerance guarantees that a decided value is either one that is proposed by an honest party or a default value $\perp$. In addition, we will require that our agreement protocol satisfies a special *validity with termination property* for up to t_s corruptions. This property ensures that if all parties input the *same value* v to the protocol, they all terminate with this value. (This property was first formalized by Blum et al. [BKL19].) Given this building block, our high-level strategy (from the view of a party P) is as follows.

- If P correctly outputs in at least $n - t_s$ broadcasts, it inputs its set of values to the intrusion-tolerant consensus protocol Π_{IT}. Otherwise, it inputs a default value $\perp'$.
- If a set of values is decided upon by Π_{IT}, P continues with the protocol. Otherwise, P participates in an execution of an asynchronous DKG protocol with $O(\lambda n^3)$ complexity with reconstruction threshold $t_s + 1$; the protocol of Das et al. [DYX+22] satisfies these requirements.

In synchrony, by the security of broadcast, all parties will propose the same set to Π_{IT} and, by the validity of consensus Π_{IT} under t_s corruptions, this set will be decided. In this case, parties do not fall back to the asynchronous path and can cheaply agree on a t_s-sharing of a field element x.

In asynchrony, the synchronous path might fail. In this case, however, agreement and intrusion tolerance of Π_{IT} ensure that all parties securely continue

[6]This discussion omits a minor technical detail: the adversary must not be able to broadcast incorrect messages on behalf of honest parties, even in asynchrony. Ensuring this, however, is easy using digital signatures.

execution of the synchronous path with the same view or collectively fall back to asynchronous DKG. In case parties do not fall back, their common view on the protocol state allows them to carry out a secure emulation of the synchronous protocol path. In either scenario, parties agree on a t_s-sharing of a field element x, *even if the network behaves asynchronously.* The following paragraphs describe how, for each phase of our protocol, we manage to keep communication below $O(\lambda n^3)$.

An Efficient Broadcast Protocol. The first ingredient we propose is an efficient multivalued synchronous broadcast protocol assuming $t < (1 - \epsilon) \cdot n$ corruptions for any constant $\epsilon \in (0, 1)$. Tsimos, Loss and Papamanthou [TLP22] propose an efficient *binary* broadcast protocol, BulletinBC, that is statically secure. BulletinBC requires $O(\lambda^2 n^2)$ communication, and is very similar to the classic Dolev-Strong protocol [DS83] except to reduce communication complexity, parties *gossip* instead of *multicast* sets of signatures during the protocol. We modify this protocol and an extension protocol from Nayak et al. [NRS+20] in Appendix B to build synchronous broadcast with $O(n\ell + \lambda n^2)$ complexity.

We use our extension protocol in the first phase of DKG. More precisely, each party P_i broadcasts 1) n Pedersen commitments corresponding to random polynomials $f_i(\cdot)$ and $f'_i(\cdot)$; 2) n ciphertexts corresponding to each party P_j's share of P_i's secret, namely $f_i(j)$ and $f'_i(j)$; and 3) n NIZK proofs that proves ciphertext j, for each $j \in [1, n]$, contains encryptions of values $f_i(j)$ and $f'_i(j)$. This obviates the need for a complaint management protocol as each party can determine the well-formedness of each message broadcast themselves. With $O(\lambda)$-sized NIZKs [PHGR13,CGG+20], each party invokes broadcast with an $O(\lambda n)$-sized message, and consequently this step incurs $O(\lambda n^3)$ communication.

Our Intrusion-Tolerant Consensus Protocol. We adapt a multivalued agreement protocol from Mostéfaoui and Raynal [MR17] to ensure intrusion tolerance and validity under t_s corruptions in synchrony. We show in Appendix C that the protocol uses $O(\ell n^3 + \lambda n^3)$ communication complexity. As such, parties can not simply propose their $O(\lambda n^2)$-sized set of sharings to consensus within DKG without incurring super-cubic complexity.

Efficiently Reconstructing the Final Output. To keep the communication complexity below $O(\lambda n^3)$, we observe that each party does not require the entire contents of the $O(\lambda n^2)$-sized set to reconstruct their share and the public key of the final secret. To this end, a party accumulates n 'personalised' values, one per party and each of size $O(\lambda n)$, into a value z that they propose to consensus. An accumulated value for party P_i contains a description of the qualified parties Q, the $|Q|$ ciphertexts P_i needs to reconstruct their share of the secret $\sum_{q \in Q} f_q(i)$ (alongside $\sum_{q \in Q} f'_q(i)$), and a Pedersen commitment corresponding to these two summations. By intrusion-tolerance, if a non-trivial value is decided by consensus, the honest party (or parties) who proposed such a z can send the relevant part and proof of membership in z to each party. By using an accumulator with accumulated value z of size at most $O(n\lambda)$ (typically we have $O(\lambda)$ [BP97,Lip12]), we thus achieve $O(\lambda n^3)$ complexity for this step. We emphasise that without the intrusion-tolerance property that it would be possible for parties to decide a value from consensus that does not correspond to an 'honest' accumulation value z. One could bypass intrusion tolerance using a consensus protocol with $O(n\ell + \lambda n^3)$ complexity that ensures *external validity* on decided values [CKPS01]. However, it appears difficult to design such a protocol using erasure codes (as is typical) as parties cannot feasibly evaluate an external validity function on a message until it is reconstructed.

From each party's personalized value, they can reconstruct their share of the secret key but not yet the public key. To reconstruct the public key, it is tempting to replace the reconstruction phase of $\mathsf{New\text{-}DKG}$ with another round of broadcast and agreement. However, this would allow an adversary to bias the distribution of the shared secret by deciding to fallback to asynchronous DKG depending on e.g., the first bit of the reconstructed public key. We therefore avoid this by publicly reconstructing the public key using the approach of Shrestha et al. in [SBKN21]. More precisely, each party P_i computes and multicasts the value $G = g^{\sum_{q \in Q} f_q(i)}$ and their accumulated value that they prove is consistent with their Pedersen commitment via an efficient Fiat-Shamir based NIZK [CGJ+99,SBKN21]. Parties can thus collect $t_s + 1$ valid points in the exponent of g and then reconstruct the public key by Lagrange interpolation in the exponent and terminate.

1.3 Technical Overview: MPC

Our starting point is the protocol by Blum, Liu-Zhang and Loss [BLL20], which gave a network-agnostic MPC given an initial setup for threshold additive-homomorphic encryption.

The protocol is composed of two parts. First, a synchronous MPC with t_s-full security when the network is synchronous, and achieves t_a-agreement on output (where the output can either be correct or $\bot$) when the network is asynchronous. Second, a purely asynchronous MPC with full security resilient to up to t_a corruptions.

The bottleneck for the communication complexity lies in the first protocol, since it requires the usage of n network-agnostic Byzantine agreement (BA) protocols per multiplication gate. Given that the most efficient network-agnostic BA protocol [DHLZ21] incurs quadratic communication, the total communication amounts to $O(n^3|C|\lambda + \mathsf{poly}(n, \lambda))$ bits. However, the most communication-efficient MPC in the purely asynchronous setting (in the same setting, from additive-homomorphic encryption) incurs $O(n^2|C|\lambda + \mathsf{poly}(n, \lambda))$ communication.

In order to decrease a linear factor in the communication, we optimize the protocol using the well-known offline-online paradigm [Bea92]. The offline phase generates ℓ Beaver triples with network-agnostic security, where ℓ is the number of multiplication gates in the circuit: if the network is synchronous and there are up to t_s corruptions, all parties output the same ℓ encrypted random multiplication triples, with plaintexts unknown to the adversary; and if the network is asynchronous and there are up to t_a corruptions, each party outputs either ℓ triples as above, or $\perp$. With these triples, one can use standard techniques to achieve an online phase with quadratic communication, where each multiplication gate is reduced to two public reconstructions [Bea92].

The protocol makes use of a number of primitives, including 1) an efficient synchronous broadcast protocol for long messages with weak-validity and 2) a network-agnostic Byzantine agreement protocol among others. In a simplified form[7], the protocol works as follows:

– Each party P_i generates ℓ random encryptions $A_i^1, \ldots, A_i^\ell$, and broadcasts them using the broadcast for long messages.
– Parties agree on a subset S of parties that received the encryptions using n instances of network-agnostic BA. If the set has size less than $n - t_s$, output $\perp$ and terminate.
– The parties compute ℓ ciphertexts, where each ciphertext is the sum of all ciphertexts coming from parties in S, i.e. $A^j = \sum_{k \in S} A_k^j$.
– Each party P_i generates ℓ random encryptions $B_i^1, \ldots, B_i^\ell$, and ciphertexts $C_i^1, \ldots, C_i^\ell$ where $C_i^j = b_i^j \cdot A^i$ and b_i^j is the plaintext of B_i^j, and broadcasts all these values using the broadcast for long messages.
– Again, parties agree on a subset S' of parties that received the encryptions, as in Step 2.
– Compute $B^j = \sum_{k \in S'} B_k^j$ and $C^j = \sum_{k \in S'} C_k^j$.
– Output the triples (A^j, B^j, C^j) for $j = 1, \ldots, \ell$.

The communication complexity amounts to n instances of broadcast (note that the cost of the BA instances is independent of the number of multiplication gates). Since each broadcast incurs $O(n\ell + \lambda n^2)$ bits of communication, the total communication is $O(n^2\ell + \lambda n^3)$, or $O(n^2)$ per generated triple (ignoring additive terms).

Intuitively, the protocol generates random triples because each component contains the contribution of at least an honest party. If the network is synchronous, all honest parties output the generated triples. However, if the network is asynchronous, some of the honest parties may not obtain the triples and output $\perp$. This will be enough in the online phase and is handled similarly as the protocol in [BLL20]).

Finally, using the DKG protocol from above, and the observation that the NIZK proofs can be generated with no setup using the multi-string honest majority proof system by Groth and Ostrovsky [GO07], we can base our MPC protocol from plain PKI.

1.4 Related Work

DKG. Many synchronous DKG protocols assume the existence of broadcast channels, i.e., that essentially abstract away secure broadcast and consensus,

[7]The simplified description tolerates only fail-stop corruptions. To achieve security against active adversaries, one needs NIZKs at appropriate steps of the protocol. See Section 4 for details.

including the seminal protocol of Gennaro et al. [GJKR07]. In a recent work, Shrestha et al. [SBKN21] consider when broadcast is no longer assumed (as in our work), and propose a protocol with $O(\lambda n^3)$ complexity which is the state-of-the-art. Canetti et al. [CGJ$^+$99] propose an adaptively-secure DKG protocol, but almost all other work, including ours, consider static security.

Das et al. [DYX$^+$22] propose an asynchronous DKG protocol with $O(\lambda n^3)$ communication complexity. In order to bypass the need for direct coin flipping, which incurs $O(\lambda n^3)$ overhead, they perform a clever reduction to n instances of binary consensus which uses $O(\lambda n^2)$ for coin flips from honest parties. Their technique relies on asynchronous complete secret sharing and is not straightforward to adapt to the network-agnostic setting. Abraham et al. use a so-called aggregatable DKG protocol [GJM$^+$21] to also build a protocol with $O(\lambda n^3)$ overhead that only requires an efficient Byzantine agreement primitive. One can likely adopt this approach to the network-agnostic setting by optimising an protocol. However, the only efficient construction of aggregatable DKG we are aware of allows parties to agree on a shared *group* element as a secret which can thus be applied only to less standard cryptosystems.

Communication complexity in MPC. The literature in communication complexity is extensive, so we are only able to cover a part of it.

In the synchronous model, solutions with linear communication, i.e. $O(\lambda n)$ bits per multiplication gate, have been known for a while (see e.g. [HN06,DI06,BTH08,BFO12,GLS19,GSZ20]), for several settings: $t < n/3$ without setup and $t < n/2$ with setup, as well as cryptographic and information-theoretic.

In the asynchronous model, information-theoretic solutions with optimal resilience $t < n/3$ were provided by Ben-Or et al. [BKR94], and later improved by Patra et al. [PCR10,PCR08] to $O(\lambda n^5)$ per multiplication, and by Choudhury [Cho20] to $O(\lambda n^4)$ per multiplication. Solutions with suboptimal resilience $t < n/4$ were achieved with linear communication $O(\lambda n)$ [SR00,PSR02,CHP13,PCR15]. For cryptographic security and optimal resilience $t < n/3$, current solutions require trusted setup, typically in the form of threshold cryptosystems. The works by [HNP05,HNP08,CHLZ21] make use of additive threshold homomorphic encryption, with the protocols [HNP08,CHLZ21] communicating $O(\lambda n^2)$ per multiplication. The work by Choudhury and Patra [CP15] achieves $O(\lambda n)$ per multiplication at the cost of using somewhat-homomorphic encryption, and the work by Cohen [Coh16] achieves communication independent of the circuit size using fully-homomorphic encryption.

In the setting with network-agnostic security, the protocols [BLL20,DHLZ21] achieve optimal resilience $t_a + 2 \cdot t_s < n$ and cryptographic security, with the first being more communication-efficient with $O(\lambda n^3 |C|)$ bits per multiplication gate (using the network-agnostic BA [DHLZ21]). The recent protocol [ACC22] achieves optimal resilience $t_a + 3 \cdot t_s < n$ in the setting of perfect security, and has communication complexity $O(\lambda n^4 |C|)$.

1.5 Paper Organisation

In Section 2, we define our model and relevant cryptographic and distributed primitives. In Section 3, we present our DKG protocol and argue for its security. In Section 4, we present our MPC protocol. In the appendices, we first define some security notions for cryptographic primitives (Appendix A). We then describe and prove secure our secure broadcast protocols (Appendix B), our intrusion-tolerant consensus protocol, defining some building blocks along the way (Appendix C), and a binary consensus protocol it requires (Appendix D), proofs deferred from the main body (Appendix E) and a figure for a synchronous MPC algorithm with unanimous output in asynchrony (Figures 14 and 15).

2 Preliminaries and Definitions

Throughout the paper, we consider a network of n parties $P_1, \ldots, P_n$ that communicate over point-to-point authenticated channels. Some fraction of these parties are controlled by an adversary and may deviate arbitrarily from the protocol. We call the uncorrupted parties honest and the corrupted parties dishonest. When we say that a party *multicasts a message*, we mean that it sends it to all n parties in the network. We denote the security parameter by λ and the random variable X output by some probabilistic experiment Π by $X \leftarrow \Pi$. We denote the set of integers from a to b by $[a, b]$. For an element x in a set S, $x \leftarrow S$ denotes x being sampled from S uniformly at random. We sometimes use *maps* or key-value stores, which are data structures of the form $\mathsf{map}[k] = v$ for lookup key k which outputs value v.

We assume that global parameters $par = (\mathbb{G}, p, g, h)$ are fixed and known to all parties. Here, $\mathbb{G}$ is a cyclic group of prime order p with independent generators g and h. Given par that does not contain h, we can choose h appropriately as, e.g., $H(1)$ where H is a random oracle of the form $H : \{0,1\}^* \to \mathbb{G}^*\mathbb{G} \setminus \{1\}$.

Public Key Infrastructure. We assume that the parties have established a public key infrastructure before the protocol execution, which is a bulletin board or plain PKI. Namely, each party P_i has an encryption/decryption key pair $\mathsf{ek}_i/\mathsf{dk}_i$ for a public-key encryption scheme and a signing/verification key pair $\mathsf{sk}_i/\mathsf{vk}_i$ for a signature scheme, where ek_i and vk_i are known to all parties. We do not assume that these keys are computed in a trusted manner and instead we assume only that each party generates them locally and then makes the public components known to everybody using a public bulletin board. In particular, malicious parties may choose their keys arbitrarily, corrupt honest parties after seeing they generate their keys and choose keys maliciously based on keys registered by honest parties. We define the function $\mathcal{VK}(P')$ callable by each party which takes as input a sequence of parties $P' = (P_{i(1)}, \ldots, P_{i(k)})$ and outputs the corresponding registered verification keys as a sequence $(\mathsf{vk}_{i(1)}, \ldots, \mathsf{vk}_{i(k)})$.

Communication Model. Our network has two possible states, the synchronous and the asynchronous state. When the network is synchronous, all parties begin the protocol at the same time, the clocks of the parties progress at the same rate, and all messages are delivered within some known finite time $\Delta > 0$ (called the network delay) after being sent. In particular, messages of honest parties can not be dropped from the network and are always delivered. Thus, we can consider protocols that execute in rounds of length Δ where parties start executing round r at time $(r-1)\Delta$. When the network is asynchronous, the adversary can delay messages arbitrarily as long as the messages exchanged between honest parties are eventually delivered. In contrast to the synchronous model, parties may start the protocol at different times in an asynchronous network, since their clocks and processing speeds are not necessarily synchronized. Finally, honest parties do not know a priori in which type of network they are in.

Adversarial Model. We assume a probabilistic polynomial-time adversary that can corrupt up to t parties. The adversary may cause the corrupted parties to deviate from the protocol arbitrarily. Furthermore, we assume a rushing adversary who may obtain messages sent to it before choosing and sending messages of its own. Moreover, we assume a static adversary, who chooses which parties to immediately corrupt before protocol execution begins.

2.1 Cryptographic Primitives

Definitions and properties that we introduce hereafter are only required to hold with probability $1 - \mathsf{negl}(\lambda)$. We defer formal definitions of correctness and security to Appendix A. We first introduce a standard definition for public-key encryption.

Definition 1 (Public-key encryption (PKE)). *A public key encryption scheme is a tuple of PPT algorithms* $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *such that:*

- KeyGen*: This is a key generation protocol that takes as input the security parameter λ. It outputs a public-secret key pair* $(\mathsf{ek}, \mathsf{dk})$*, denoted as* $(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{KeyGen}(\lambda)$.
- Enc*: This is a probabilistic encryption algorithm that takes as input a public key* ek *and a message $m \in \{0,1\}^*$ (a bit string). It outputs a ciphertext c, denoted as* $c \leftarrow \mathsf{Enc}(\mathsf{ek}, m)$.
- Dec*: This is a deterministic decryption algorithm that takes as input a decryption key* dk *and a ciphertext c. It outputs a message m, denoted as* $m \leftarrow \mathsf{Dec}(\mathsf{dk}, c)$*, where possibly $m = \perp$ denoting failure.*

Next, we define non-interactive zero-knowledge proofs (NIZKs). NIZKs enable a prover to non-interactively (i.e., generate a message that is then verified) to prove to a verifier the validity of a statement without revealing anything else.

Definition 2 (Non-interactive zero-knowledge proof (NIZK) [Gro06]). *Let R be an NP relation and L the corresponding language. A non-interactive zero-knowledge proof is a tuple of PPT algorithms* $(\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify})$ *such that:*

- Gen*: This is a parameter generation algorithm that takes as input the security parameter λ. It outputs parameters par.*

– Prove*: This is a probabilistic proving algorithm that takes as input a statement X to be proven and the corresponding witness w where $(X, w) \in L$. It outputs a proof π, denoted as $\pi \leftarrow \mathsf{Prove}(X, w)$.*
– Verify*: This is a deterministic verification algorithm that takes as input a statement X and a proof π. It outputs an acceptance bit b, denoted as $b \leftarrow \mathsf{Verify}(X, \pi)$.*

We assume that NIZKs are of size $O(\lambda)$. The NIZKs that we use in our DKG construction can be constructed using efficient, Fiat-Shamir style proofs in the random oracle model [SBKN21,CGG$^+$20]. Alternatively, one can use SNARKs with a common reference string setup [PHGR13].

We then define accumulators, a primitive that enables a party to accumulate several values from some set D into an accumulated value z. At this point, the party can generate (compact) proofs that verify that a given value is in D. A secure accumulator is in particular one where 'invalid' proofs are hard to forge.

Definition 3 (Cryptographic accumulator). *A cryptographic accumulator is a tuple of PPT algorithms* $(\mathsf{Gen}, \mathsf{Eval}, \mathsf{CreateWit}, \mathsf{Verify})$ *such that:*

– Gen*: This is an accumulator key generation algorithm that takes as input the security parameter λ and an accumulation threshold n. It outputs a (public) accumulator key ak.*
– Eval*: This is a deterministic evaluation algorithm that takes as input an accumulator key ak and a set $D = \{d_1, \ldots, d_n\}$ to be accumulated. It outputs an accumulation value z for D, denoted as $z \leftarrow \mathsf{Eval}(ak, D)$.*
– CreateWit*: This is a probabilistic witness creation algorithm that takes as input an accumulator key ak, an accumulation value z for D, and a value d_i. It outputs $\perp$ if $d_i \notin D$, and a witness w_i otherwise, denoted as $w_i \leftarrow \mathsf{CreateWit}(ak, z, d_i)$.*
– Verify*: This takes as input an accumulator key ak, an accumulation value z for D, a witness w_i, and a value d_i. It outputs an acceptance bit b, denoted as $b \leftarrow \mathsf{Verify}(ak, z, w_i, d_i)$, where $b = 1$ when w_i proves that $d_i \in D$.*

The helper function CreateWits*, denoted as $(w_1, \ldots, w_n) \leftarrow \mathsf{CreateWits}(ak, z, D)$ for set $D = \{d_1, \ldots, d_n\}$, is shorthand for the n calls $(\mathsf{CreateWit}(ak, z, d_1), \ldots, \mathsf{CreateWit}(ak, z, d_n))$.*

Note that the above definition does not consider updates or removals of elements from the accumulated value z, and so our definition is weaker than that of much of the literature. We require an accumulator with witnesses w and accumulation values z of size $O(\lambda)$. We also require that operations after ak was generated are *deterministic*. The classic RSA accumulator satisfies these requirements with trusted setup in the standard model [BP97]; without trusted setup, one can use, for instance, the accumulator from Lipmaa in [Lip12]. Looking further, it can be seen that our protocols can use vector commitments instead of accumulators, which can also be built without trusted setup with constant-sized openings [CF13].

Definition 4 (Linear erasure codes). *A Reed-Solomon (RS) code [RS60] is a linear error correction code in the finite field $\mathbb{F}_{2^a}$, parameterized by n and b with $n \leq 2^a - 1$, given by the tuple of algorithms* $(\mathsf{Encode}, \mathsf{Decode})$ *such that:*

– Encode*: This is an encoding algorithm that takes as input b data symbols $(m_1, \ldots, m_b) \in \mathbb{F}_{2^a}^b$ and outputs a codeword $(s_1, \ldots, s_n) \in \mathbb{F}_{2^a}^n$ of length n, denoted as $(s_1, \ldots, s_n) \leftarrow \mathsf{Encode}(m_1, \ldots, m_b)$. Knowledge of any b elements of the codeword uniquely determines the input message and the rest of the codeword.*
– Decode*: This is a decoding algorithm that takes as input a codeword $(s_1, \ldots, s_n)$ of length n and outputs b symbols $(m_1, \ldots, m_b) \in \mathbb{F}_{2^a}^b$, denoted as $(m_1, \ldots, m_b) \leftarrow \mathsf{Decode}(s_1, \ldots, s_n)$. It can tolerate up to c errors and d erasures in codewords $(s_1, \ldots, s_n)$ if and only if $n - b \geq 2c + d$.*

Our protocol that uses erasure codes will use $b = n - t$. Finally, we assume that parties have a threshold additively homomorphic encryption setup available. That is, it provides to each party P_i a global public key ek and a private key share dk_i.

Definition 5 (Threshold homomorphic encryption). *A threshold homomorphic encryption scheme is a tuple of PPT algorithms* $(\mathsf{Keygen}, \mathsf{TEnc}, \mathsf{TDec})$ *such that:*

– Keygen*: This key generation algorithm takes as input integers (t, n) and outputs key pair $(\mathsf{ek}, \mathsf{dk})$, where ek is the public key, and $\mathsf{dk} = (\mathsf{dk}_1, \ldots, \mathsf{dk}_n)$ is the list of private keys, denoted as $(\mathsf{ek}, \mathsf{dk}) = \mathsf{Keygen}_{(t,n)}(1^\lambda)$.*
– TEnc*: This takes as input an encryption key ek and plaintext m and outputs an encryption $\mathsf{TEnc}_{\mathsf{ek}}(m)$ of m, which we denote explicitly.*
– TDec*: Given a ciphertext c and a secret key share dk_i, there is an algorithm that outputs $d_i = \mathsf{TDec}_{\mathsf{dk}_i}(c)$, such that $(d_1, \ldots, d_n)$ forms a t-out-of-n sharing of the plaintext $m = \mathsf{Dec}_{\mathsf{dk}}(c)$. Moreover, with t decryption shares $\{d_i\}$, one can reconstruct the plaintext $m = \mathsf{TRec}(\{d_i\})$.*

It further satisfies the following properties:

– Additively homomorphic*: Given ek and two encryptions $\mathsf{Enc}_{\mathsf{ek}}(a)$ and $\mathsf{TEnc}_{\mathsf{ek}}(b)$, one can efficiently compute an encryption $\mathsf{Enc}_{\mathsf{ek}}(a + b)$.*
– Multiplication by constant*: Given ek, a plaintext α and an encryption $\mathsf{Enc}_{\mathsf{ek}}(a)$, one can efficiently compute a random encryption $\mathsf{Enc}_{\mathsf{ek}}(\alpha a)$.*

Such a threshold encryption scheme can be based on, for example, the Paillier cryptosystem [Pai99].

2.2 Distributed Primitives

When relevant, our primitives take input from a value set V with $|V| \geq 2$; we assume that default value $\perp \notin V$. We distinguish between algorithms that *generate output* (generally called liveness), and algorithms that additionally *terminate*. In particular, an algorithm may be live but not terminating, since it may need to still remain online and send more messages to help other parties output. Our treatment of liveness and termination varies between the primitives we introduce below. Note that $\perp$ is considered as a valid output in each protocol.

We first introduce intrusion-tolerant Byzantine agreement and secure broadcast, the two main building blocks we use to build DKG. Byzantine agreement is a classic primitive that allows parties which each input a value to agree on a common output value. We define liveness (generating output) and termination in two separate properties below. We emphasise that our definition captures the standard Byzantine agreement problem.

Definition 6 (Byzantine agreement). *Let Π be a protocol executed by parties $P_1, \ldots, P_n$, where each party P_i begins holding input $v_i \in V$.*

- **Validity:** *Π is t-*valid *if the following holds whenever at most t parties are corrupted: if every honest party's input is equal to the same value v, then every honest party outputs v.*
- **Consistency:** *Π is t-*consistent *if whenever at most t parties are corrupted, every honest party that outputs a value outputs the same value v.*
- **Liveness:** *Π is t-*live *if whenever at most t parties are corrupted, every honest party outputs a value $v \in V \cup \{\perp\}$.*
- **Termination:** *Π is t-*terminating *if whenever at most t parties are corrupted, every honest party terminates.*
- **Intrusion tolerance:** *Π is t-*intrusion tolerant *if whenever at most t parties are corrupted, every honest party that outputs a value either outputs an honest party's input v or $\perp$.*
- **Validity with termination:** *Π is t-*valid with termination *if the following holds whenever at most t parties are corrupted: if every honest party's input is equal to the same value v, then every honest party outputs v and terminates.*

*If Π is t-valid, t-consistent, t-live, and t-terminating, we say it is t-*secure*. If Π is t-secure and is t-intrusion tolerant, we say it is t-*secure with intrusion tolerance*.*

In secure broadcast (or just broadcast), parties aim to agree on a value which is either the value chosen by the designated sender or a default value (in case the sender is corrupted). Our definition handles termination directly, even in asynchrony (where we only guarantee weak validity). As for Byzantine agreement, the following captures the standard broadcast primitive.

Definition 7 (Secure broadcast (BC)). *Let Π be a protocol executed by parties $P_1, \ldots, P_n$, where a designated party P begins holding input $v \in V$.*

- **Validity:** *Π is t-*valid *if whenever at most t parties are corrupted: if party P is honest and inputs v, then all honest parties P_j output $v = v'$.*
- **Consistency:** *Π is t-*consistent *if whenever at most t parties are corrupted, every honest party outputs the same value v'.*
- **Liveness:** *Π is t-*live *if whenever at most t parties are corrupted, every honest party outputs a value $v' \in V \cup \{\perp\}$.*

- **Termination:** *Π is t-*terminating *if whenever at most t parties are corrupted, every honest party terminates.*
- **External validity:** *Π is t-*externally valid *if the following holds whenever at most t parties are corrupted: if honest party P_i outputs v', then for validity predicate Q, $Q(v)$ is true.*
- **Weak validity:** *Π is t-*weakly valid *if whenever at most t parties are corrupted: if P is honest and inputs v, then all honest parties P_i output either v or $\bot$ and terminate upon generating output.*

*If Π is t-valid, t-consistent, t-live and t-terminating, we say it is t-*secure*.*

Note that weak validity was defined in [BKL19] and external validity was introduced in [CKPS01] for Byzantine agreement.

We define our distributed key generation (DKG) primitive. In DKG, a set of parties collaborates to share a uniformly random secret. Each party outputs the public key corresponding to the secret, their own secret share and a set of public shares that parties can use to prove ownership of their share. We restrict our definition to the case where parties share a uniform *field element* associated to some group generated by g, i.e. a public key $y = g^x$ and secret x; one can generalise or vary the definition to capture other settings.

Definition 8 (Distributed key generation (DKG)). *Let Π be a protocol executed by parties $P_1, \ldots, P_n$, where each party P_i outputs a secret key share ss_i, a vector of public key shares $(\mathsf{ps}_1, \ldots, \mathsf{ps}_n)$, a public key pk and parties terminate upon generating output.*

- **Correctness:** *Π is (t, d)-*correct *for $d > t$ if whenever at most t parties are corrupted, there exists a polynomial $f \in \mathbb{Z}_p[X]$ of degree $d - 1$ such that for all $i \in [1, n]$, $\mathsf{ss}_i = f(i)$ and $\mathsf{ps}_i = g^{\mathsf{ss}_i}$. Moreover, $\mathsf{pk} = g^{f(0)}$.*
- **Consistency:** *Π is t-*consistent *if whenever at most t parties are corrupted, all honest parties output the same public key pk and the same vector of public key shares $(\mathsf{ps}_1, \ldots, \mathsf{ps}_n)$.*
- **Secrecy:** *Π is t-*secret *if the following holds whenever at most t parties are corrupted: For every (PPT) adversary $\mathcal{A}$, there exists a (PPT) simulator $\mathcal{S}$ with the following property. On input an element $y \in \mathbb{G}$ and a set of corrupted parties B with $|B| \leq t$, $\mathcal{S}$ generates a transcript whose distribution is computationally indistinguishable from $\mathcal{A}$'s view of a run of Π with corrupted set B in which all honest parties output y as their public key.*
- **Uniformity:** *Π is t-*uniform *if the following holds whenever at most t parties are corrupted: Fix $y \in \mathbb{G}$. Then, for every (PPT) adversary $\mathcal{A}$, for every honest party that outputs public key pk, $\mathsf{pk} = y$ holds with probability negligibly close to $1/p$, where the probability is taken over $\mathcal{A}$'s randomness (and not the coins used in setup).*

*If Π is (t, d)-correct, t-consistent, t-secret, and t-uniform, we say it is (t, d)-*secure*.*

Our definition is adapted from that of Bacho and Loss [BL22] except we only require a standard secrecy notion akin to that of Gennaro et al. [GJKR07]. As we consider static security, our simulator is parametrised by the set of corrupted parties B chosen by the adversary. Apart from our additional uniformity property, the main difference is that we allow the secret threshold to be a value d that exceeds the number of corruptions t by more than 1. Looking forward, our DKG protocol will satisfy (t_s, d)-security in synchrony and (t_a, d)-security in asynchrony for $d = t_s + 1$. In particular, our protocol achieves t_a-secrecy in asynchrony. The definition of secrecy is not well-defined in asynchrony when considering more than t_a corruptions, because in particular not all parties may output y (or worse yet they may output different keys). One could define a variant of secrecy that guarantees 'secrecy with abort' but its usefulness is less clear given only a subset of honest parties could output a secret share.

2.3 Multi-Party Computation

A multi-party computation (MPC) protocol allows n parties $P_1, \dots, P_n$, where each party P_i has a private input x_i, to jointly compute a function over the inputs $f(x_1, \dots, x_n)$ in such a way that nothing beyond the output is revealed.

Different levels of security guarantees have been considered in the MPC literature, such as guaranteed output delivery (a.k.a. full security), where honest parties are guaranteed to obtain the correct output, or security with selective abort [IOZ14,CL17], where the adversary can choose any subset of parties to receive $\perp$, instead of the correct output. In the case of *unanimous* abort [GMW87,FGH$^+$02], the adversary can choose whether all honest parties receive the correct output or all honest parties receive $\perp$ as output.

When the network is asynchronous, it is provably impossible that the computed function takes into account all inputs from honest parties [BCG93,BKR94], since one cannot distinguish between a dishonest party not sending its input, or an honest party's input being delayed. Hence, we say that a protocol achieves L-output quality, if the output to be computed contains the inputs from at least L parties. This is modeled in the ideal functionality as allowing the ideal adversary to choose a subset S of L parties. The functionality then computes $f(x_1, \dots, x_n)$, where $x_i = v_i$ is the input of P_i in the case that $P_i \in S$, and otherwise $x_i = \perp$.

We describe the ideal functionality $\mathcal{F}_{\mathsf{sfe}}^{\mathsf{sec},L}$ for MPC with full security and L-output quality below.

In addition, we denote the functionality $\mathcal{F}_{\mathsf{sfe}}^{\mathtt{sout},L}$ (resp. $\mathcal{F}_{\mathsf{sfe}}^{\mathtt{uout},L}$), the above functionality, where the adversary can selectively choose any subset of parties to obtain $\perp$ as the output (resp. choose that either all honest parties receive $f(x_1, \dots, x_n)$ or $\perp$).

Definition 9. *A protocol π achieves full security (resp. selective abort; unanimous abort) with L output-quality if it UC-realizes functionality $\mathcal{F}_{\mathsf{sfe}}^{\mathsf{sec},L}$ ($\mathcal{F}_{\mathsf{sfe}}^{\mathtt{sout},L}$; $\mathcal{F}_{\mathsf{sfe}}^{\mathtt{uout},L}$).*

$\mathcal{F}_{\mathsf{sfe}}^{\mathsf{sec},L}$

- $\mathcal{F}_{\mathsf{sfe}}$ is parameterized by a set $\mathcal{P}$ of n parties and a function $f : (\{0,1\}^* \cup \{\bot\})^n \to (\{0,1\}^*)^n$. For each $P_i \in \mathcal{P}$, initialize the variables $x_i = y_i = \bot$. Set $S = \mathcal{P}$.
- On input (Input, v) from $P_i \in \mathcal{P}$, if $P_i \in S$, set $x_i = v$ and send a message (Input, P_i) to the adversary.
- On input $(\mathsf{OutputSet}, S')$ from the ideal adversary, where $S' \subseteq \mathcal{P}$ and $|S'| = L$, set $S = S'$ and $x_i = \bot$ for each $P_i \notin S$.
- Once all inputs from honest parties in S have been input, set each $y_i = f(x_1, \ldots, x_n)$.
- On input $(\mathsf{GetOutput})$ from P_i, output $(\mathsf{Output}, y_i, \mathsf{sid})$ to P_i.

Fig. 1: Secure Function Evaluation Functionality.

Since protocols run in a synchronous network typically achieve n-output quality, we implicitly assume that all synchronous protocols we discuss achieve n-output quality (unless otherwise specified).

Weak termination. In this work, similar to that of [BLL20], we consider protocols with the following weaker termination property: we say that a protocol has weak termination, if parties are guaranteed to terminate upon receiving an output different than $\bot$, but do not necessarily terminate if the output is $\bot$.

3 Communication-Efficient Network-Agnostic DKG

In this section, we construct our distributed key generation protocol $\Pi_{\mathsf{DKG}}^{t_a,t_s}$ with threshold $d = t_s + 1$ which we will show is t_s- (resp. t_a-) secure when run over a synchronous (resp. asynchronous) network. We present Π_{DKG} in Figure 2 which uses two helper functions that we define separately in Figure 3. We recall public parameters $par = (\mathbb{G}, p, g, h)$ introduced in Section 2, where g and h are independent generators of group $\mathbb{G}$ of prime order p. Π_{DKG} relies on the following underlying protocols:

- Π_{ADKG}, an asynchronous DKG protocol. We assume that Π_{ADKG} is (t_a, d)-secure with threshold $d = t_s + 1$ and has $O(\lambda n^3)$ communication complexity. The protocol from Das et al. [DYX+22] satisfies these requirements.
- $\Pi_{\mathsf{BC\text{-}Ext}}$, a secure broadcast protocol with default value $\bot_{bc}$. We assume that $\Pi_{\mathsf{BC\text{-}Ext}}$ is t_s-secure when run on a synchronous network, t_a-weakly valid on an asynchronous network, t_s-externally valid and t_s-intrusion tolerant. For a message of length ℓ, we require that $\Pi_{\mathsf{BC\text{-}Ext}}$ has communication complexity $O(\ell n + \lambda n^2)$. The protocol $\Pi_{\mathsf{BC\text{-}Ext}}$ defined in Figure 7 satisfies these requirements.
- Π_{IT}, a multivalued Byzantine agreement protocol with default value $\bot_{it}$. We assume that Π_{IT} is t_a-secure with intrusion tolerance and t_s-valid with

$\mathit{\Pi}_{\mathsf{DKG}}^{t_a,t_s}$

1. Set $\mathsf{it} = \mathsf{bc} = \mathsf{ready} = \mathsf{false}$. Choose two random polynomials $f_i(z)$, $f'_i(z)$ over $\mathbb{Z}_p$ of degree t_s $f_i(z) = a_{i0} + a_{i1}z + \ldots + a_{it_s}z^{t_s}$, $f'_i(z) = b_{i0} + b_{i1}z + \ldots + b_{it_s}z^{t_s}$. Let $z_i = a_{i0}$. Then:
   (a) For all $k \in [0, t_s]$, compute $C_{ik} = g^{a_{ik}}h^{b_{ik}}$.
   (b) For all $j \in [1, n]$, compute $s_{ij} = f_i(j)$, $u_{ij} = f'_i(j)$ and $c_{ij} = \mathsf{pke.Enc}(\mathsf{ek}_j, (s_{ij}, u_{ij}))$.
   (c) For all $j \in [1, n]$, compute $\pi_{ij} = \mathsf{nizk}_1.\mathsf{Prove}(X_1, w_1 = (s_{ij}, u_{ij}))$.

   Let $M_i = (C_i = (C_{i0}, \ldots, C_{it_s}), c_i = (c_{i1}, \ldots, c_{in}), \pi_i = (\pi_{i1}, \ldots, \pi_{in}))$. Run n instances of $\mathit{\Pi}_{\mathsf{BC\text{-}Ext}}^{t_s,\frac{1}{2}}$ with senders $P_1, \ldots, P_n$ using external validity predicate Valid with input M_i in instance i.
2. At time T: Let $(M'_1, \ldots, M'_n)$ be the output from the n instances of $\mathit{\Pi}_{\mathsf{BC\text{-}Ext}}^{t_s,\frac{1}{2}}$ where M'_j was output by instance $j \in [1, n]$:
   (a) If $\left|\{j \in [1, n] : M'_j \neq \perp_{bc}\}\right| \geq n - t_s$: Compute $(L_1, \ldots, L_n) = \mathsf{Split}(M'_1, \ldots, M'_n)$, $\mathsf{acc.Eval}(ak, (L_1, \ldots, L_n)) = z$ and $(w_1, \ldots, w_n) = \mathsf{acc.CreateWits}(ak, z, ((1, L_1), \ldots, (n, L_n)))$. Set $\mathsf{bc} = \mathsf{true}$ and run $\mathit{\Pi}_{\mathsf{IT}}^{t_a,t_s}$ using input z.
   (b) Otherwise, set $\mathsf{bc} = \mathsf{true}$ and run $\mathit{\Pi}_{\mathsf{IT}}^{t_a,t_s}$ using input $\perp_{dkg}$.
3. Upon receiving output z_{it} from $\mathit{\Pi}_{\mathsf{IT}}^{t_a,t_s}$: if $z_{it} \in \{\perp_{it}, \perp_{dkg}\}$, run $\mathit{\Pi}_{\mathsf{ADKG}}$. Otherwise, set $\mathsf{it} = \mathsf{true}$.
4. Upon setting $\mathsf{it} = \mathsf{bc} = \mathsf{true}$: if $z = z_{it}$, for $j \in [1, n]$, send $(\mathsf{part}, L_j, w_j)$ to P_j.
5. Upon receiving $(\mathsf{part}, L_i = (c_i^*, C_i^*, Q), w_i)$: wait until $\mathsf{it} = \mathsf{true}$ (never satisfied if $\mathit{\Pi}_{\mathsf{ADKG}}$ is invoked). Then, if $z_{it} \neq \perp$ and $\mathsf{acc.Verify}(ak, z_{it}, w_i, (i, L_i)) = 1$, wait until $\mathsf{bc} = \mathsf{true}$ and then set $\mathsf{ready} = \mathsf{true}$.
6. Upon setting $\mathsf{ready} = \mathsf{true}$: let $x_i = \sum_{j \in Q} s_{ji}$ and $x'_i = \sum_{j \in Q} u_{ji}$ where s_{ji} and u_{ji} are decrypted from c_i^* using $\mathsf{pke.Dec}$. Compute $D_i = g^{x_i}$, $C''_i = g^{x_i}h^{x'_i}$ and $\pi'_i = \mathsf{nizk}_2.\mathsf{Prove}(X_2, w_2 = (x_i, x'_i))$ where X_2 is defined with respect to $A = D_i$ and $B = C''_i$. Multicast $(\mathsf{recon}, D_i, \pi'_j, L_j = (c_j^*, C_j^*, Q), w_j)$.
7. Upon receiving $(\mathsf{recon}, D_j, \pi'_j, L_j = (c_j^*, C_j^*, Q), w_j)$ for $t_s + 1$ distinct values j with 1) $\mathsf{acc.Verify}(ak, z_{it}, w_j, (j, L_j)) = 1$ and 2) $\mathsf{nizk}_2.\mathsf{Verify}(X_2, \pi'_i) = 1$ using $A = D_j$ and $B = C_j^*$:
   (a) Wait until $\mathsf{ready} = \mathsf{true}$.
   (b) Consider $g^{F(x)}$, where $F(x)$ is the polynomial defined by Lagrange interpolation of the $t_s + 1$ distinct values $D_j = g^{x_j} = g^{F(j)}$ in the exponent. Compute $y = g^{F(0)}$. For $j \in [1, n]$, compute $\mathsf{ps}_j = g^{F(j)}$. Output $(x_i, (\mathsf{ps}_1, \ldots, \mathsf{ps}_n), y)$ and terminate.

Fig. 2: DKG protocol with threshold $d = t_s + 1$ from the perspective of party P_i. Note that under synchrony that each step will be executed in sequence.

Helper functions for $\Pi_{\mathsf{DKG}}^{t_a,t_s}$

- $\underline{\mathsf{Valid}(M_j)}$: Return true if and only M_j can be parsed as $M_j = (C_i = (C_{j0}, \ldots, C_{jt_s}), c_j = (c_{j1}, \ldots, c_{jn}), \pi_j = (\pi_{j1}, \ldots, \pi_{jn}))$ and, for all $k \in [1, n]$, $\mathsf{nizk}_1.\mathsf{Verify}(X_1, \pi_{jk}) = 1$.
- $\underline{\mathsf{Split}(M_1, \ldots, M_n)}$: Let $Q = [q_1, \ldots, q_s] \subseteq [1, n]$ be the maximal set such that $M_j \neq \perp$. Then, for $j \in [1, n]$: let $c_j^* = (c_{q_1 j}, \ldots, c_{q_s j})$ and $C_j^* = \prod_{q \in Q} C_{j,q}^*$ where $C_{j,q}^* = \prod_{k=0}^{t_s} (C_{mk})^{j^k}$ for each $q \in Q$, and then let $L_j = (c_j^*, C_j^*, Q)$. Finally, output $(L_1, \ldots, L_n)$.

Fig. 3: DKG helper functions from the perspective of party P_i.

termination, and has $O(\lambda n^3)$ communication complexity. The protocol Π_{IT} defined in Figure 11 satisfies these requirements.

We also assume the existence of a public-key encryption scheme pke, an accumulator acc and a linear erasure coding scheme rs. Finally, we require two NIZKs, nizk_1 and nizk_2, which define the following relations:

- nizk_1: Statements X_1 and witnesses $(s_{ij}, u_{ij}) \in \mathbb{Z}_p^2$, where X_1 is the statement that $\prod_{k=0}^{t_s} (C_{ik})^{j^k} = g^{s_{ij}} h^{u_{ij}}$ and c_{ij} is an encryption of (s_{ij}, u_{ij}) under ek_j, where variables C_{ik} and c_{ij} are introduced in step 1 of Π_{DKG}.
- nizk_2: Statements X_2 and witnesses $(x_i, x_i') \in \mathbb{Z}_p^2$, where X_2 is the statement, given (public) values A and B, that $A = g^{x_i}$ and $B = g^{x_i} h^{x_i'}$.

Π_{DKG} proceeds as follows.

Step 1: Let P_i be an honest party executing Π_{DKG}. P_i chooses two random polynomials $f_i(\cdot)$, $f_i(\cdot)'$ of degree t_s with coefficients a_{ik} and b_{ik} for $k \in [0, t_s]$. In this step, P_i will share points $(j, f_i(j))$ and $(j, f_i(j)')$ with each party P_j, $j \in [1, n]$, using public-key encryption scheme pke. As in Pedersen's verifiable secret sharing scheme [Ped92], P_i will also compute Pedersen commitments $C_{ik} = g^{a_{ik}} h^{b_{ik}}$ that allow parties to evaluate the polynomials in the exponents g and h together. In particular, the inclusion of polynomial f' blinds f such that values that contribute to the final secret are hidden from the adversary until after it has been decided, preventing the adversary from biasing the secret. In order for all parties to verify that all parties have received correct sharings, P_i will further compute a NIZK π_{ij} via nizk_1 for each P_j that verifies that the encrypted values under P_j's key are exactly $f_i(j)$ and $f_i(j)'$. All n parties then invoke $\Pi_{\mathsf{BC\text{-}Ext}}$ (secure broadcast), inputting a message to the i-th instance containing these Pedersen commitments, encryptions for all n parties and the corresponding NIZK proofs.

Steps 2 and 3: If the network is synchronous, then by t_s-security of $\Pi_{\mathsf{BC\text{-}Ext}}$ and since at least $n - t_s$ honest parties broadcast, all parties will agree on the same set of values of size $\geq n - t_s$ once all instances of $\Pi_{\mathsf{BC\text{-}Ext}}$ terminate at the same time T. By t_s-external validity of $\mathit{\Pi}_{\mathsf{BC\text{-}Ext}}$, only messages that are Valid (Figure 3) – namely, those which are well-formed and contain n valid NIZKs – can be output. Note in asynchrony that $\mathit{\Pi}_{\mathsf{BC\text{-}Ext}}$ does not satisfy consistency, so honest parties could output different messages. To resolve this, it would be natural for parties to execute consensus on the output of $\mathit{\Pi}_{\mathsf{BC\text{-}Ext}}$ that ensures t_s-validity in synchrony and t_a-security in asynchrony. However, not all parties may output $n - t_s$ values from $\mathit{\Pi}_{\mathsf{BC\text{-}Ext}}$, so parties require a mechanism to 'abort' if not enough values are obtained from consensus.

We use intrusion-tolerant consensus $\mathit{\Pi}_{\mathsf{IT}}$ to efficiently solve this problem. Rather than proposing the entire, $O(\lambda n^2)$-sized output of $\mathit{\Pi}_{\mathsf{BC\text{-}Ext}}$ to consensus, P_i instead proposes an accumulated value z to $\mathit{\Pi}_{\mathsf{IT}}$. Intuitively, z accumulates n values (one per party) each of size $O(\lambda n)$ corresponding to the information that each party 'needs' to eventually reconstruct their secret share and the common public key; we describe these values further below. If an honest party does not output enough values from $\mathit{\Pi}_{\mathsf{BC\text{-}Ext}}$, they instead propose $\bot_{dkg}$ to $\mathit{\Pi}_{\mathsf{IT}}$. $\mathit{\Pi}_{\mathsf{IT}}$ guarantees that a decided value is either one proposed by an honest party or $\bot$. Consequently, if $\mathit{\Pi}_{\mathsf{IT}}$ outputs $v \in \{\bot, \bot_{dkg}\}$, all honest parties fallback to $\mathit{\Pi}_{\mathsf{ADKG}}$. This will not occur in synchrony and may or may not occur in asynchrony. Otherwise, all honest parties output the same accumulated value z.

<u>Steps 4 and 5:</u> If $z \notin \{\bot, \bot_{dkg}\}$ is decided by $\mathit{\Pi}_{\mathsf{IT}}$, then z must have been proposed by an honest party, say P_j. Assuming this is true, P_j (plus any other honest party that output z) sends each party their 'value' accumulated in z alongside a proof of membership. Party P_i obtains their value L_i this way, where L_i is computed using Split (Figure 3). More precisely, L_i contains:

- The same Q for all n parties, corresponding to the 'qualified' set of parties of size $\geq n - t_s$ from which P_j received values from $\mathit{\Pi}_{\mathsf{BC\text{-}Ext}}$;
- $|Q|$ ciphertexts encrypting $f_q(i)$ and $f'_q(i)$ to P_i for all $q \in Q$; and
- Commitment $C^*_j = g^{\sum_{q\in Q} f_q(i)} h^{\sum_{q\in Q} f'_q(i)}$.

These messages allow each party to reconstruct a sharing of a secret $\sum_{q\in Q} f_q(0)$. After deciding z from $\mathit{\Pi}_{\mathsf{IT}}$, P_j sends the relevant part message to all parties, which parties verify is correct with $\mathsf{acc.Verify}$.

<u>Steps 6 and 7:</u> At this point, P_i has received a valid message of the form $(\mathsf{part}, L_i = (c^*_i, C^*_i, Q), w_i)$. By decrypting values in c^*_i, P_i can deduce its own secret share $x_i = \sum_{j\in Q} f_j(i)$ but not necessarily the corresponding public shares $g^{F(1)}, \ldots, g^{F(n)}$ and public key $g^{F(0)}$. Thus, parties will collaborate to compute g^x by reconstructing the polynomial $F(\cdot) = \sum_{j\in Q} f_j(\cdot)$ in the exponent of g. To this end, parties will reveal their share g^{x_i} and then compute a proof with nizk_2 that shows that it is consistent with the sharings of polynomials $f_j(\cdot)$ and $f'_j(\cdot)$ in step 1 of the protocol (which were previously hidden). More precisely, P_i computes $D_i = g^{x_i}$, $x'_i = \sum_{j\in Q} f'_j(i)$ (by decryption of c^*_i), $C''_i = g^{x_i} h^{x'_i}$ and $\pi'_j = \mathsf{nizk}_2.\mathsf{Prove}(X_2, (x_i, x'_i))$. Then, P_i multicasts a reconstruction message

recon containing D_i, the proof π'_j and P_i's value L_i alongside w_i, the proof of inclusion in z.

On receipt of a recon message from P_j, P_i can verify that 1) L_j was accumulated in z (using $\mathsf{acc.Verify}$), and 2) the NIZK π'_j is correct and, in particular, is consistent with the value $C_i^* = g^{x_i^*} h^{x_i'^*}$ contained in L_j. Because these checks pass, the value C_i^* must be of the form $g^{x_i} h^{x'_i}$ computed by a honest party that output z from Π_{IT}, and thus the value D_j contained in the recon message must be of the form $g^{\sum_{k \in Q} f_k(j)}$, i.e. it must be a valid share. When P_i receives $t_s + 1$ such values, P_i evaluates $F(0)$ in the exponent of g to derive public key g^x and $F(j)$ for $j \in [1, n]$ to derive the n public shares. At this point, P_i terminates.

Communication Complexity. At step 1, each party invokes secure broadcast with $O(\lambda n)$-sized input (assuming NIZKs are size $O(\lambda)$, each which costs $O(n\ell + \lambda n^2)$, so this step incurs $O(\lambda n^3)$ overhead. Apart from using generic NIZKs, one can instantiate nizk_1 with $O(\lambda)$-sized proofs in a suitable Paillier group under the decisional composite residuosity assumption [CGG+20]. At step 2, Π_{IT} takes $O(\lambda n^3)$ communication. If parties invoke Π_{ADKG}, then steps 4 to 7 are ignored, and Π_{ADKG} costs $O(\lambda n^3)$ itself. At step 4, $O(n)$ parties send n part messages, each of size $O(\lambda n)$, so this incurs $O(\lambda n^3)$ overhead. At step 5, $O(n)$ parties multicast a recon message of size $O(\lambda n)$, incurring $O(\lambda n^3)$ overhead, again assuming nizk_2 has $O(\lambda)$-sized proofs. nizk_2 can be instantiated using the efficient NIZK used in [SBKN21] in the random oracle model in any cryptographic group $\mathbb{G}$. Thus, Π_{DKG} has a communication complexity of $O(\lambda n^3)$.

We defer proofs for results stated in the main body hereafter to Appendix E.

Theorem 1. *Let n, t_s, t_a be such that $0 \le t_a < \frac{n}{3} \le t_s < \frac{n}{2}$ and $t_a + 2 \cdot t_s < n$, and let $d = t_s + 1$. Then distributed key generation protocol $\Pi_{\mathsf{DKG}}^{t_a, t_s}$ (Figures 2 and 3) is (t_s, d)-secure when run on a synchronous network and (t_a, d)-secure when run on an asynchronous network.*

Corruption Thresholds. Our construction shows that $t_a + 2 \cdot t_s < n$ corruptions are sufficient to ensure (t_s, d)-security in synchrony and (t_a, d)-security in asynchrony for $d = t_s + 1$. We also note that it is also *necessary*:

Lemma 1. *Let n, t_a, t_s be such that $t_a + 2 \cdot t_s \ge n$. Then, if DKG protocol Π is t_s-uniform in a synchronous network, then it cannot also be t_a-consistent in an asynchronous network.*

4 Multi-Party Computation with Asynchronous Fallback Guarantees

4.1 Multi-Party Zero-Knowledge Protocols

Let us assume a binary relation R, consisting of pairs (x, w), where x is the statement, and w is a witness to the statement. A zero-knowledge proof allows a prover P to prove to a verifier V knowledge of w such that $R(x, w) = 1$. We are interested in zero-knowledge proofs for three types of relations, parameterized by a threshold encryption scheme with public encryption key ek:

1. *Proof of Plaintext Knowledge:* The statement consists of ek, and a ciphertext c. The witness consists of a plaintext m and randomness r such that $c = \mathsf{TEnc}_{\mathsf{ek}}(m, r)$.
2. *Proof of Correct Multiplication:* The statement consists of ek, and ciphertexts c_1, c_2 and c_3. The witness consists of a plaintext m_1 and randomness r_1, r_3 such that $c_1 = \mathsf{TEnc}_{\mathsf{ek}}(m_1, r_1)$ and $c_3 = m_1 \cdot c_2 + \mathsf{TEnc}_{\mathsf{ek}}(0; r_3)$.
3. *Proof of Correct Decryption:* The statement consists of ek, a ciphertext c, and a decryption share d. The witness consists of a decryption key share dk_i, such that $d = \mathsf{TDec}_{\mathsf{dk}_i}(c)$.

Assuming a PKI infrastructure and honest majority, one can realize a Non-interactive Zero-Knowledge Proof (NIZK) system (without the need to assume a trusted CRS setup) using the multi-string honest majority NIZK by Groth and Ostrovsky [GO07]. By distributing the NIZK proof with a (multivalued) secure broadcast protocol $\Pi_{\mathsf{BC}}^{t_a,t_s}$ that is t_s-secure in synchrony and t_a-weak-validity when the network is asynchronous, we obtain the following lemma:

Lemma 2. *Let R be a relation. Let n, t_s, t_a be such that $t_a, t_s < n/2$. Assuming honest majority, there is a protocol that realizes the multi-party zero-knowledge functionality for P as prover with the following guarantees:*

1. *When run in a synchronous network, it achieves full security up to t_s corruptions.*
2. *When run in an asynchronous network, it achieves security with selective abort up to t_a corruptions.*

4.2 Protocol Description

We describe an optimized version of the MPC protocol with fallback by Blum, Liu-Zhang and Loss [BLL20], with communication complexity $O(n^2\lambda)$ bits per multiplication gate. This matches the asymptotic communication complexity of the current most efficient purely asynchronous MPC protocols [HNP08,CHLZ21] in the setting of optimal resilience $t < n/3$, without the use of multiplicative-homomorphic threshold encryption schemes.

The protocol makes use of a threshold additive homomorphic encryption scheme $(\mathsf{Keygen}, \mathsf{TEnc}, \mathsf{TDec}, \mathsf{TRec})$ and a secure broadcast protocol $\Pi_{\mathsf{BC}}^{t_s,1/2}$ that achieves t_s-security when the network is synchronous and t_a-weak validity when the network is asynchronous and with communication complexity $O(n\ell + \mathsf{poly}(n, \lambda))$, where ℓ is the input size.

The protocol is divided into two phases: an offline and an online phase. The offline phase generates Beaver multiplication triples (in encrypted form) and can be executed without the knowledge of the inputs. In the online phase, parties distribute their inputs and process the circuit to evaluate in a gate-by-gate fashion, where addition gates are processed locally and multiplication gates are processed with the help of the Beaver triples, via two public reconstructions.

Triple Generation. In order to generate Beaver triples (Figure 4), we make use of a multi-valued broadcast protocol $\Pi_{\mathsf{BC}}^{t_s,1/2}$ that is t_s-secure when run on a synchronous network and t_a-weakly valid that terminates after T_{bc} rounds.

$\Pi_{\mathsf{triples}}^{t_a,t_s}(\ell)$

- Each P_j computes random values $a_j^1, \ldots, a_j^\ell$ and computes corresponding encryptions $A_j^1, \ldots, A_j^\ell$, where $A_j^i = \mathsf{TEnc}(a_j^i)$. Then, use the multi-valued broadcast protocol with weak-validity $\Pi_{\mathsf{BC}}^{t_s,1/2}$ to broadcast all these values. Also use the multi-party zero-knowledge $\mathcal{F}_{\mathsf{mzk}}$ to prove plaintext knowledge of the ciphertexts. Wait for T_{bc} time.
- Let S_i be the subset of the parties succeeding with the proof towards party P_i. Run n times the protocol $\Pi_{\mathsf{BA}}^{t_a,t_s}$, each one to decide for each party P_j's proof. Input 1 to party j's BA if and only if $j \in S_i$. Wait for T_{ba}.
- Let S be the subset of the parties for which $\Pi_{\mathsf{BA}}^{t_a,t_s}$ outputs 1. Check that $|S| \geq n - t_s$. If not, output $\perp$ and terminate.
- Compute $A^i = \sum_{k \in S} A_k^i$.
- Each P_j computes random values $b_j^1, \ldots, b_j^\ell$ and computes corresponding encryptions $B_j^1, \ldots, B_j^\ell$, where $B_j^i = \mathsf{TEnc}(b_j^i)$ and $C_j^i = b_j^i \cdot A^i$. Then, use the multi-valued broadcast protocol $\Pi_{\mathsf{BC}}^{t_s,1/2}$ to broadcast all these values. Also use the multi-party zero-knowledge $\mathcal{F}_{\mathsf{mzk}}$ to give proofs of correct multiplication for each (B_j^i, A^i, C_j^i). Wait for T_{bc} time.
- Let S_i' be the subset of the parties succeeding with the proof towards party P_i. Run n times the protocol $\Pi_{\mathsf{BA}}^{t_a,t_s}$, each one to decide for each party P_j's proof. Input 1 to party j's BA if and only if $j \in S_i$. Wait for T_{ba}.
- Let S' be the subset of the parties for which $\Pi_{\mathsf{BA}}^{t_a,t_s}$ outputs 1. Check that $|S'| \geq n - t_s$. If not, output $\perp$ and terminate.
- Compute $B^i = \sum_{k \in S'} B_k^i$ and $C^i = \sum_{k \in S'} C_k^i$.
- Output the triples (A^i, B^i, C^i) for each $i = 1 \ldots, \ell$.

Fig. 4: Beaver Triple Generation.

Communication Complexity. The communication complexity amounts to n parallel instances of secure broadcast with input size ℓ encryptions and non-interactive zero-knowledge proofs, and an additive term (independent of the number ℓ) corresponding to n parallel instances of BA. This incurs a total communication of $O(\lambda n^2 \ell + \mathsf{poly}(n, \lambda))$ bits.

Lemma 3. *Let n, t_s, t_a be such that $t_a, t_s < n$. $\Pi_{\mathsf{triples}}^{t_a,t_s}(\ell)$ is an n-party protocol with communication complexity $O(n^2 \ell + \mathsf{poly}(n, \lambda))$, achieving the following guarantees:*

- *When the network is synchronous and there are up to t_s corruptions, all parties output the same ℓ encrypted random multiplication triples, with the plaintexts unknown to the adversary.*

– *When the network is asynchronous and there are up to t_a corruptions, the output of each party P_i is either ℓ encrypted random multiplication triples with the plaintexts unknown to the adversary or $\perp$.*

Synchronous Protocol with Unanimous Output. We present the synchronous MPC protocol that achieves full security when the network is synchronous and there are t_s corruptions, but also achieves unanimous output up to t_a corruptions under an asynchronous network. The protocol is an optimized version of the one in [BLL20], where the multiplication gates are executed using Beaver triples generated during an Offline Phase, and incurs a communication complexity of $O(n^2)$ field elements per multiplication gate. We defer the protocol description to the appendices (Figures 14 and 15), and only provide a high level description here.

The protocol closely follows the one by Blum, Liu-Zhang and Loss [BLL20], which uses a setup for threshold additive-homomorphic encryption. This approach was initially introduced by Cramer, Damgard and Nielsen [CDN01], and the idea is that parties keep threshold encryptions of the circuit wires and perform computations on a gate-by-gate fashion. First, the inputs are distributed in the form of a threshold encryption. Since the threshold encryption scheme is additively homomorphic, the addition gates can be performed locally by the parties. Multiplication gates are processed in a standard manner using Beaver triples. The only difference in the network-agnostic setting is that in some parts of the protocol (such as the input distribution, or the triples generation), in the case the network is asynchronous, there might be information missing (e.g. input ciphertexts or encrypted triples). For that, the protocol in [BLL20] makes use of an abort flag. As soon as a party detects that not enough information has arrived by a certain amount of time, it sets the flag to 1, and stops executing further steps of the protocol. This can only make the protocol stall, but will not compromise security. Before the output is decrypted, an agreement on a core-set sub-primitive also known as ACS (see [BLL20] for details on this primitive) is run to see whether parties must decrypt or not. This ensures that parties agree on whether the output was computed. If yes, they can jointly (and safely) decrypt the output ciphertext. If not, all parties output $\perp$.

Lemma 4. *Let n, t_s, t_a be such that $0 \leq t_a < n/3 \leq t_s < n/2$ and $t_a + 2t_s < n$. Protocol $\Pi_{\mathsf{smpc}}^{t_s,t_a}$ has communication complexity $O(\lambda n^2|C| + \mathsf{poly}(n, \lambda))$ bits, where the C is the circuit to evaluate, and satisfies:*

– *When run in a synchronous network, it achieves full security up to t_s corruptions.*
– *When run in an asynchronous network, it achieves unanimous output with weak termination up to t_a corruptions and has $n - t_s$ output quality.*

4.3 Protocol Compiler

In this section, we restate the protocol $\Pi_{\mathsf{mpc}}^{t_s,t_a}$ for secure function evaluation presented in [BLL20] which tolerates up to t_s (resp. t_a) corruptions when the network is synchronous (resp. asynchronous), for any $0 \leq t_a < \frac{n}{3} \leq t_s < \frac{n}{2}$ satisfying $t_a + 2t_s < n$. The protocol is based on two sub-protocols:

– $\Pi_{\mathsf{smpc}}^{t_s,t_a}$ is a secure function evaluation protocol which gives full security up to t_s corruptions when run in a synchronous network, and achieves unanimous output with weak termination up to t_a corruptions and has $n - t_s$ output quality when run in an asynchronous network.
– $\Pi_{\mathsf{ampc}}^{t_a}$ is a secure function evaluation protocol which gives full security up to t_a corruptions and has $n - t_a$ output quality when run in an asynchronous network.

Theorem 2 ([BLL20]). *Let n, t_s, t_a be such that $0 \leq t_a < \frac{n}{3} \leq t_s < \frac{n}{2}$ and $t_a + 2t_s < n$. Given sub-protocols $\Pi_{\mathsf{smpc}}^{t_s,t_a}$ and $\Pi_{\mathsf{ampc}}^{t_a}$ with the guarantees described above, there is a protocol $\Pi_{\mathsf{mpc}}^{t_s,t_a}$ with communication complexity the sum of the communication of the two sub-protocols, satisfying the following properties:*

1. *When run in a synchronous network, it achieves full security up to t_s corruptions.*
2. *When run in an asynchronous network, it achieves full security up to t_a corruptions and has $n - t_s$ output quality.*

Using Lemma 4 and Theorem 2, and a quadratic asynchronous protocol (see e.g. [HNP08]) we obtain a protocol $\Pi_{\mathsf{mpc}}^{t_s,t_a}$ with communication complexity $O(n^2)$ field elements per multiplication gate. Moreover, using Theorem 1, we can base our protocol on a plain public-key infrastructure.

Corollary 1. *Assuming a plain PKI, there is an MPC protocol $\Pi_{\mathsf{mpc}}^{t_s,t_a}$ with communication complexity $O(n^2|C|\lambda + \mathsf{poly}(n, \lambda))$ bits, satisfying the following properties:*

1. *When run in a synchronous network, it achieves full security up to t_s corruptions.*
2. *When run in an asynchronous network, it achieves full security up to t_a corruptions and has $n - t_s$ output quality.*

References

ABKL22. Andreea B. Alexandru, Erica Blum, Jonathan Katz, and Julian Loss. State machine replication under changing network conditions. Cryptology ePrint Archive, Paper 2022/698, 2022. `https://eprint.iacr.org/2022/698`.

ACC22. Ananya Appan, Anirudh Chandramouli, and Ashish Choudhury. Perfectly-secure synchronous mpc with asynchronous fallback guarantees. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*, PODC'22, page 92–102, New York, NY, USA, 2022. Association for Computing Machinery.

AJM$^+$21. Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. Reaching consensus for asynchronous distributed key generation. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, pages 363–373, 2021.

BCG93. Michael Ben-Or, Ran Canetti, and Oded Goldreich. Asynchronous secure computation. In *25th ACM STOC*, pages 52–61. ACM Press, May 1993.

Bea92. Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 420–432. Springer, Heidelberg, August 1992.

BFO12. Eli Ben-Sasson, Serge Fehr, and Rafail Ostrovsky. Near-linear unconditionally-secure multiparty computation with a dishonest minority. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 663–680. Springer, Heidelberg, August 2012.

BGLS03. Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 416–432. Springer, Heidelberg, May 2003.

BKL19. Erica Blum, Jonathan Katz, and Julian Loss. Synchronous consensus with optimal asynchronous fallback guarantees. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part I*, volume 11891 of *LNCS*, pages 131–150. Springer, Heidelberg, December 2019.

BKL21. Erica Blum, Jonathan Katz, and Julian Loss. Tardigrade: An atomic broadcast protocol for arbitrary network conditions. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part II*, volume 13091 of *LNCS*, pages 547–572. Springer, Heidelberg, December 2021.

BKR94. Michael Ben-Or, Boaz Kelmer, and Tal Rabin. Asynchronous secure computations with optimal resilience (extended abstract). In Jim Anderson and Sam Toueg, editors, *13th ACM PODC*, pages 183–192. ACM, August 1994.

BL22. Renas Bacho and Julian Loss. On the adaptive security of the threshold bls signature scheme. To appear at ACM CCS, 2022.

BLL20. Erica Blum, Chen-Da Liu-Zhang, and Julian Loss. Always have a backup plan: Fully secure synchronous MPC with asynchronous fallback. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 707–731. Springer, Heidelberg, August 2020.

BP97. Niko Bari and Birgit Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 480–494. Springer, Heidelberg, May 1997.

BTH08. Zuzana Beerliová-Trubíniová and Martin Hirt. Perfectly-secure MPC with linear communication complexity. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 213–230. Springer, Heidelberg, March 2008.

CDN01. Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. Multiparty computation from threshold homomorphic encryption. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 280–299. Springer, Heidelberg, May 2001.

CF13. Dario Catalano and Dario Fiore. Vector commitments and their applications. In *International Workshop on Public Key Cryptography*, pages 55–72. Springer, 2013.

CGG$^+$20. Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. UC non-interactive, proactive, threshold ECDSA with identifiable aborts. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1769–1787. ACM Press, November 2020.

CGJ+99. Ran Canetti, Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Adaptive security for threshold cryptosystems. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 98–115. Springer, Heidelberg, August 1999.

CGMA85. Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*, pages 383–395. IEEE, 1985.

CHLZ21. Annick Chopard, Martin Hirt, and Chen-Da Liu-Zhang. On communication-efficient asynchronous MPC with adaptive security. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part II*, volume 13043 of *LNCS*, pages 35–65. Springer, Heidelberg, November 2021.

Cho20. Ashish Choudhury. Optimally-resilient unconditionally-secure asynchronous multi-party computation revisited. Cryptology ePrint Archive, Report 2020/906, 2020. `https://eprint.iacr.org/2020/906`.

CHP13. Ashish Choudhury, Martin Hirt, and Arpita Patra. Asynchronous multiparty computation with linear communication complexity. In Yehuda Afek, editor, *Distributed Computing*, pages 388–402, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

CKPS01. Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In *Annual International Cryptology Conference*, pages 524–541. Springer, 2001.

CL17. Ran Cohen and Yehuda Lindell. Fairness versus guaranteed output delivery in secure multiparty computation. *Journal of Cryptology*, 30(4):1157–1186, October 2017.

Coh16. Ran Cohen. Asynchronous secure multiparty computation in constant time. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016, Part II*, volume 9615 of *LNCS*, pages 183–207. Springer, Heidelberg, March 2016.

CP15. Ashish Choudhury and Arpita Patra. Optimally resilient asynchronous mpc with linear communication complexity. In *Proceedings of the 2015 International Conference on Distributed Computing and Networking*, ICDCN '15, New York, NY, USA, 2015. Association for Computing Machinery.

DHLZ21. Giovanni Deligios, Martin Hirt, and Chen-Da Liu-Zhang. Round-efficient byzantine agreement and multi-party computation with asynchronous fallback. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part I*, volume 13042 of *LNCS*, pages 623–653. Springer, Heidelberg, November 2021.

DI06. Ivan Damgård and Yuval Ishai. Scalable secure multiparty computation. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 501–520. Springer, Heidelberg, August 2006.

DS83. Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.

DYX+22. Sourav Das, Thomas Yurek, Zhuolun Xiang, Andrew Miller, Lefteris Kokoris-Kogias, and Ling Ren. Practical asynchronous distributed key generation. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 2518–2534, 2022.

Fel87. Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 427–438. IEEE, 1987.

FGH$^+$02. Matthias Fitzi, Daniel Gottesman, Martin Hirt, Thomas Holenstein, and Adam Smith. Detectable byzantine agreement secure against faulty majorities. In Aleta Ricciardi, editor, *21st ACM PODC*, pages 118–126. ACM, July 2002.

GJKR99. Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 295–310. Springer, Heidelberg, May 1999.

GJKR07. Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 20(1):51–83, January 2007.

GJM$^+$21. Kobi Gurkan, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. Aggregatable distributed key generation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 147–176. Springer, 2021.

GLL$^+$21. Yingzi Gao, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. Efficient asynchronous byzantine agreement without private setups. *arXiv preprint arXiv:2106.07831*, 2021.

GLS19. Vipul Goyal, Yanyi Liu, and Yifan Song. Communication-efficient unconditional MPC with guaranteed output delivery. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part II*, volume 11693 of *LNCS*, pages 85–114. Springer, Heidelberg, August 2019.

GMW87. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

GO07. Jens Groth and Rafail Ostrovsky. Cryptography in the multi-string model. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 323–341. Springer, Heidelberg, August 2007.

Gro06. Jens Groth. Simulation-sound NIZK proofs for a practical language and constant size group signatures. In Xuejia Lai and Kefei Chen, editors, *ASIACRYPT 2006*, volume 4284 of *LNCS*, pages 444–459. Springer, Heidelberg, December 2006.

GSZ20. Vipul Goyal, Yifan Song, and Chenzhi Zhu. Guaranteed output delivery comes free in honest majority MPC. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 618–646. Springer, Heidelberg, August 2020.

HN06. Martin Hirt and Jesper Buus Nielsen. Robust multiparty computation with linear communication complexity. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 463–482. Springer, Heidelberg, August 2006.

HNP05. Martin Hirt, Jesper Buus Nielsen, and Bartosz Przydatek. Cryptographic asynchronous multi-party computation with optimal resilience (extended abstract). In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 322–340. Springer, Heidelberg, May 2005.

HNP08. Martin Hirt, Jesper Buus Nielsen, and Bartosz Przydatek. Asynchronous multi-party computation with quadratic communication. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 473–485. Springer, Heidelberg, July 2008.

IOZ14. Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. Secure multi-party computation with identifiable abort. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 369–386. Springer, Heidelberg, August 2014.

KG09. Aniket Kate and Ian Goldberg. Distributed key generation for the internet. In *2009 29th IEEE International Conference on Distributed Computing Systems*, pages 119–128. IEEE, 2009.

Lip12. Helger Lipmaa. Secure accumulators from euclidean rings without trusted setup. In Feng Bao, Pierangela Samarati, and Jianying Zhou, editors, *ACNS 12*, volume 7341 of *LNCS*, pages 224–240. Springer, Heidelberg, June 2012.

MR10. Achour Mostéfaoui and Michel Raynal. Signature-free broadcast-based intrusion tolerance: never decide a byzantine value. In *International Conference On Principles Of Distributed Systems*, pages 143–158. Springer, 2010.

MR17. Achour Mostéfaoui and Michel Raynal. Signature-free asynchronous byzantine systems: from multivalued to binary consensus with $t < n/3$, $O(n^2)$ messages, and constant time. *Acta Informatica*, 54(5):501–520, 2017.

NRS$^+$20. Kartik Nayak, Ling Ren, Elaine Shi, Nitin H. Vaidya, and Zhuolun Xiang. Improved Extension Protocols for Byzantine Broadcast and Agreement. In Hagit Attiya, editor, *34th International Symposium on Distributed Computing (DISC 2020)*, volume 179 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 28:1–28:17, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

Pai99. Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238. Springer, Heidelberg, May 1999.

PCR08. Arpita Patra, Ashish Choudhury, and C. Pandu Rangan. Efficient asynchronous multiparty computation with optimal resilience. Cryptology ePrint Archive, Report 2008/425, 2008. `https://eprint.iacr.org/2008/425`.

PCR10. Arpita Patra, Ashish Choudhary, and C. Pandu Rangan. Efficient statistical asynchronous verifiable secret sharing with optimal resilience. In Kaoru Kurosawa, editor, *ICITS 09*, volume 5973 of *LNCS*, pages 74–92. Springer, Heidelberg, December 2010.

PCR15. Arpita Patra, Ashish Choudhury, and C. Pandu Rangan. Efficient asynchronous verifiable secret sharing and multiparty computation. *Journal of Cryptology*, 28(1):49–109, January 2015.

Ped91. Torben P. Pedersen. A threshold cryptosystem without a trusted party (extended abstract) (rump session). In Donald W. Davies, editor, *EUROCRYPT'91*, volume 547 of *LNCS*, pages 522–526. Springer, Heidelberg, April 1991.

Ped92. Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 129–140. Springer, Heidelberg, August 1992.

PHGR13. Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 238–252. IEEE Computer Society Press, May 2013.

PSR02. B. Prabhu, K. Srinathan, and C. Pandu Rangan. Asynchronous unconditionally secure computation: An efficiency improvement. In Alfred Menezes and Palash Sarkar, editors, *INDOCRYPT 2002*, volume 2551 of *LNCS*, pages 93–107. Springer, Heidelberg, December 2002.

RS60. Irving S Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics*, 8(2):300–304, 1960.

SBKN21. Nibesh Shrestha, Adithya Bhat, Aniket Kate, and Kartik Nayak. Synchronous distributed key generation without broadcasts. Cryptology ePrint Archive, Paper 2021/1635, 2021. `https://eprint.iacr.org/2021/1635`.

SR00. K. Srinathan and C. Pandu Rangan. Efficient asynchronous secure multiparty distributed computation. In Bimal K. Roy and Eiji Okamoto, editors, *INDOCRYPT 2000*, volume 1977 of *LNCS*, pages 117–129. Springer, Heidelberg, December 2000.

Sta96. Markus Stadler. Publicly verifiable secret sharing. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 190–199. Springer, Heidelberg, May 1996.

TLP22. Georgios Tsimos, Julian Loss, and Charalampos Papamanthou. Gossiping for communication-efficient broadcast. Springer-Verlag, 2022.

Appendices

A Deferred Definitions and Security Notions

We first define aggregate signatures which we use later in these appendices. In an aggregate signature scheme, a party can use their signing key to sign a message individually. All parties can also call $\mathsf{Combine}$ to combine several (possibly already aggregated) signatures with respect to the same message to form a new signature on the same message. As usual, signatures can also be verified via Verify. We emphasise that $\mathsf{Combine}$ and Verify are non-interactive algorithms in this work.

Definition 10 (Aggregate signatures). *An aggregate signature scheme is a 4-tuple of PPT algorithms* $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Combine}, \mathsf{Verify})$ *such that:*

- KeyGen*: This is a key generation protocol that takes as input the security parameter* λ *and outputs* n *independent public-secret key pairs* $(\mathsf{vk}_i, \mathsf{sk}_i)$, $i \in [1, n]$.
- Sign*: This is a probabilistic signing algorithm that takes as input a secret key* sk_i *and a message* $m \in \{0,1\}^*$*. It outputs a signature* σ_i*, denoted as* $\sigma_i \leftarrow \mathsf{Sign}(\mathsf{sk}_i, m)$.
- $\mathsf{Combine}$*: This is a deterministic signature combining algorithm that takes as input a sequence of signatures* $\Sigma = (\sigma_{i(1)}, \ldots, \sigma_{i(k)})$*, the corresponding sequence of sets of verification keys* $VK = (\mathsf{vk}_{i(1)}, \ldots, \mathsf{vk}_{i(k)})$ *and a message* m*. It outputs either an aggregate signature* σ *with respect to public keys* $\cup_{j \in [1,k]} \mathsf{vk}_{i(j)}$*, denoted as* $\sigma \leftarrow \mathsf{Combine}(\Sigma, VK, m)$*, or* $\perp$.
- Verify*: This is a deterministic signature verification algorithm that takes as input a message* m*, an aggregate signature* σ*, and the set of verification keys* $VK = \{vk_1, \ldots, vk_k\}$ *corresponding to* σ*. It outputs an acceptance bit* b*, denoted as* $b \leftarrow \mathsf{Verify}(VK, \sigma, m)$*, where 1 denotes acceptance.*

We sometimes write $\langle m \rangle_i$*, which is defined as* (m, σ) *where* $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_i, m)$*. For a single verification key* vk *(resp. signature* σ*), we sometimes write* vk *(resp.* σ*) instead of* $\{\mathsf{vk}\}$ *(resp.* $\{\sigma\}$*) as input to* $\mathsf{Combine}$ *and* Verify.

One can instantiate aggregate signatures of size $O(\lambda) + P$ in the random oracle model, where P is the size of representing the signers (in our work $P = n$ using a bitmask and PKI for parties to map indices to public keys locally) [BGLS03]. Similarly, we require individual (non-aggregated) signatures of size $O(\lambda)$. We implicitly assume domain separation when signing messages in protocols we introduce.

We introduce relevant security definitions for a public key encryption scheme (PKE), an aggregate signature scheme, an cryptographic accumulator and a non-interactive zero-knowledge proof (NIZK). We begin with the definition of CPA-security of a PKE scheme.

Definition 11 (CPA-Security of PKE). *Let* $\Pi_{\mathsf{PKE}} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *be a public key encryption scheme. For* $b \in \{0,1\}$ *and an algorithm* A*, define experiment* $\mathbf{CPA}^{\mathsf{A}}_{\Pi_{\mathsf{PKE}}}(\lambda, b)$ *as follows:*

1. *Run the key generation algorithm and get* $(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{KeyGen}(\lambda)$*.*
2. *Run* A *on input* $(\mathsf{ek}, \mathsf{dk})$ *and get a pair of same-length messages* m_0, m_1*.*
3. *Compute the ciphertext* $c \leftarrow \mathsf{Enc}(\mathsf{ek}, m_b)$ *and run* A *on input* c*.*
4. *When* A *returns* $b' \in \{0,1\}$*, the experiment returns* b'*.*

We say that Π_{PKE} *has indistinguishable encryptions against chosen plaintext attacks (CPA-security) if for all PPT algorithms* A*, we have*

$$|\Pr[\mathbf{CPA}^{\mathsf{A}}_{\Pi_{\mathsf{PKE}}}(\lambda, 1) = 1] - \Pr[\mathbf{CPA}^{\mathsf{A}}_{\Pi_{\mathsf{PKE}}}(\lambda, 0) = 1]| \le \mathsf{negl}(\lambda)\,.$$

We proceed with the definition of a collision-resistant accumulator, which we see as the notion of a secure accumulator.

Definition 12 (Collision-Resistant Accumulator). *Let* $\Pi_{\mathsf{Acc}} = (\mathsf{Gen}, \mathsf{Eval}, \mathsf{CreateWit}, \mathsf{Verify})$ *be a cryptographic accumulator. For set size* n *and an algorithm* A*, define experiment* $\mathbf{CR}^{\mathsf{A}}_{\Pi_{\mathsf{Acc}}}(\lambda, n)$ *as follows:*

1. *Run the generation algorithm and get* $ak \leftarrow \mathsf{Gen}(\lambda, n)$*.*
2. *Run* A *on input* (n, ak) *and get* $(\{d_1, \ldots, d_n\}, d', w')$*.*
3. *Compute the accumulation value* $z \leftarrow \mathsf{Eval}(ak, \{d_1, \ldots, d_n\})$*.*
4. *If* $d' \notin \{d_1, \ldots, d'\}$ *and* $\mathsf{Verify}(ak, z, w', d') = 1$*, the experiment returns 1. Otherwise it returns 0.*

We say that Π_{Acc} *is collision-resistant or secure if for all PPT algorithms* A *and any* n*, we have* $\Pr[\mathbf{CR}^{\mathsf{A}}_{\Pi_{\mathsf{Acc}}}(\lambda, n) = 1] \le \mathsf{negl}(\lambda)$*.*

We proceed with the definition of the security of an aggregate signature scheme, which is given by the unforgeability (under chosen message attack) of aggregate signatures.

Definition 13 (Unforgeability under Chosen Message Attack). *Let* $\Pi_{\mathsf{AggSgn}} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Combine}, \mathsf{Verify})$ *be an aggregate signature scheme. For an algorithm* A*, define experiment* $\mathbf{UF\text{-}CMA}^{\mathsf{A}}_{\Pi_{\mathsf{AggSgn}}}(\lambda)$ *as follows:*

1. *Run the key generation algorithm and get public-secret key pair* $(\mathsf{vk}_i, \mathsf{sk}_i)$*.* A *is given a public key* vk_1*.*
2. *At any time of the experiment,* A *gets access to a signing oracle that answer queries of the following type: When* A *submits a message* $m \in \{0,1\}^*$ *of its choice, return* $\sigma_1 \leftarrow \mathsf{Sign}(\mathsf{sk}_1, m)$*.*
3. A *outputs* $k-1$ *additional public keys* $\mathsf{vk}_2, \ldots, \mathsf{vk}_k$ *for some* $k \ge 1$ *and a message* m^**.* A *outputs an aggregate signature* σ^* *with respect to public keys* $VK = \{\mathsf{vk}_1, \ldots, \mathsf{vk}_k\}$ *and message* m^**.*
4. *If* $\mathsf{Verify}(VK, \sigma^*, m^*) = 1$ *and* m^* *was not queried previously by* A*, the experiment returns 1. Otherwise it returns 0.*

We say that $\mathit{\Pi}_{\mathsf{AggSgn}}$ is unforgeable under chosen message attack (UF-CMA) or just secure if for all PPT algorithms A*, we have* $\Pr[\mathbf{UF\text{-}CMA}^{\mathsf{A}}_{\mathit{\Pi}_{\mathsf{AggSgn}}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$*.*

We end this section with some security notions for a non-interactive zero-knowledge proof. For this, we require perfect completeness, zero-knowledge and simulation-sound extractability. Henceforth, we let R be an NP relation and L the corresponding language.

Definition 14 (Perfect Completeness of NIZK). *Let* $\mathit{\Pi}_{\mathsf{NIZK}} = (\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify})$ *be a non-interactive zero-knowledge proof. For an algorithm* A*, define experiment* $\mathbf{PerfComp}^{\mathsf{A}}_{\mathit{\Pi}_{\mathsf{NIZK}}}(\lambda)$ *as follows:*

1. *Run the parameter generation algorithm and get* $par \leftarrow \mathsf{Gen}(\lambda)$*.*
2. *Run* A *on input* par *and get* $(X, w) \leftarrow \mathsf{A}(par)$*.*
3. *Compute the proof* $\pi \leftarrow \mathsf{Prove}(X, w)$*.*
4. *If* $(X, w) \in L$ *and* $\mathsf{Verify}(X, \pi) = 1$*, the experiment returns 1. Otherwise it returns 0.*

We say that $\mathit{\Pi}_{\mathsf{NIZK}}$ *has perfect completeness if for all algorithms* A*, we have* $\Pr[\mathbf{PerfComp}^{\mathsf{A}}_{\mathit{\Pi}_{\mathsf{NIZK}}}(\lambda) = 1] = 1$*.*

Definition 15 (Zero-Knowledge of NIZK). *Let* $\mathit{\Pi}_{\mathsf{NIZK}} = (\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify})$ *be a non-interactive zero-knowledge proof. Let* $\mathsf{S} = (\mathsf{S}_1, \mathsf{S}_2)$ *be a pair of PPT algorithms (called the simulator). Furthermore, let* $\mathsf{S}'(par, \tau, X, w) = \mathsf{S}_2(par, \tau, X)$ *if* $(X, w) \in L$ *and* $\mathsf{S}'(par, \tau, X, w) = 0$ *if* $(X, w) \notin L$*. For an algorithm* A*, we define the advantage of* A *as*

$$\begin{aligned}\mathbf{Adv\text{-}ZK}^{\mathsf{A},\mathsf{S}}_{\mathit{\Pi}_{\mathsf{NIZK}}}(\lambda) = |\Pr[&par \leftarrow \mathsf{Gen}(\lambda) : \mathsf{A}^{\mathsf{Prove}(par,\cdot,\cdot)}(par) = 1] \\ &- \Pr[(par, \tau) \leftarrow \mathsf{S}_1(\lambda) : \mathsf{A}^{\mathsf{S}'(par,\tau,\cdot,\cdot)}(par) = 1]|.\end{aligned}$$

We say that $\mathit{\Pi}_{\mathsf{NIZK}}$ *has zero-knowledge if there exists a simulator* S *as above such that for all non-uniform PPT algorithms* A*, we have* $\mathbf{Adv\text{-}ZK}^{\mathsf{A},\mathsf{S}}_{\mathit{\Pi}_{\mathsf{NIZK}}}(\lambda) \leq \mathsf{negl}(\lambda)$*.*

Definition 16 (Simulation-Soundness of NIZK). *Let* $\mathit{\Pi}_{\mathsf{NIZK}} = (\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify})$ *be a non-interactive zero-knowledge proof. Let* $\mathsf{S} = (\mathsf{S}_1, \mathsf{S}_2)$ *be a pair of PPT algorithms (called the simulator). For an algorithm* A*, we define the advantage of* A*, where* Q *is the list of simulation queries and responses, as*

$$\begin{aligned}\mathbf{Adv\text{-}ss}^{\mathsf{A},\mathsf{S}}_{\mathit{\Pi}_{\mathsf{NIZK}}}(\lambda) = \Pr[&(par, \tau) \leftarrow \mathsf{S}_1(\lambda), (X, \pi) \leftarrow \mathsf{A}^{\mathsf{S}_2(par,\tau,\cdot)}(par), \\ &w \leftarrow \mathsf{Verify}(par, X, \pi) = 1, (X, w) \notin Q \cup L].\end{aligned}$$

We say that $\mathit{\Pi}_{\mathsf{NIZK}}$ *has simulation soundness if there exists a simulator* S *as above such that for all non-uniform PPT algorithms* A*, we have* $\mathbf{Adv\text{-}ss}^{\mathsf{A},\mathsf{S}}_{\mathit{\Pi}_{\mathsf{NIZK}}}(\lambda) \leq \mathsf{negl}(\lambda)$*.*

B Communication-Efficient Synchronous Broadcast

In this section, we construct a synchronous secure broadcast protocol with complexity $O(\ell n + \lambda n^2)$ that tolerates $t < (1-\epsilon) \cdot n$ corruptions for $\epsilon \in (0,1)$ for messages of length ℓ. To do so, we adapt the extension protocol proposed by Nayak et al. [NRS⁺20]. Their protocol, however, relies on a λ-bit broadcast module with the same corruption tolerance and communication complexity $O(\lambda n^2)$. We therefore first construct such a protocol.

B.1 Short Message Broadcast Module

We present our protocol $\Pi_{\mathsf{BC}}^{t,\epsilon}$ in Figure 5 that allows λ-bit messages to be broadcast in $O(\lambda n^2)$ communication complexity. We assume the existence of an aggregate signature scheme as. Let $R = O(\log n)$ and $q = O(1/\epsilon)$ be two constants that we use in the protocol and proof below.

Our protocol is similar to $\mathsf{BulletinBC}$ from [TLP22] (Fig. 2), which is in turn similar to the well-known Dolev-Strong protocol. Whereas signatures are multicast to all users in Dolev-Strong, signatures are sent to each party with probability q/n in $\mathsf{BulletinBC}$. To ensure security, $\mathsf{BulletinBC}$ thus requires an additional $R = O(\log n)$ rounds to ensure that the 'gossiped' message propagates to all parties except with negligible probability. Notably, we extend $\mathsf{BulletinBC}$ to support *multivalued* broadcast and improve communication complexity by using aggregate signatures.

In $\Pi_{\mathsf{BC}}^{t,\epsilon}$, each party P_i manages the two local maps $\mathsf{sent}, \mathsf{detect} : M \to \{\mathsf{false}, \mathsf{true}\}$ and initially sets $\mathsf{sent}[m] = \mathsf{detect}[m] = \mathsf{false}$ for all $m \in M$. Then, the designated sender P^* multicasts its signed input value m_i and sets $\mathsf{sent}[m_i] = \mathsf{detect}[m_i] = \mathsf{true}$ (step 1).

The protocol then runs in $t + R$ rounds. In rounds $1 \le r \le t + R$, for each $m \in M$, if P_i has received a signature on m signed by the sender P^*, then if P_i can form a valid aggregate signature with $\min\{r-1, t\}$ signers,[8] and their set $\{m' \in M : \mathsf{sent}[m'] = \mathsf{true}\}$ is of size ≤ 1, P_i sets $\mathsf{sent}[m] = \mathsf{true}$, computes an aggregate signature on it and sends this plus P^*'s signature to each party with probability q/n.

Note if we simply replace the (deterministic) multicast from Dolev-Strong with probabilistic sending, then consistency may not hold if P^* signs more than two messages. In particular, messages received by honest parties may not reach all honest parties, since we require for efficiency that they do not gossip more than two messages. To deal with this, we consider P^*'s signatures separately to those of other parties. In particular, when P_i receives a signature σ of P^* on m, if $\mathsf{detect}[m] = \mathsf{false}$ and $|\{m' : \mathsf{detect}[m] = \mathsf{true}\}| < 2$, P_i *multicasts* m and the signature σ and sets $\mathsf{detect}[m] = \mathsf{true}$.[9]

[8]For rounds $r \ge t+1$ we do not require r signatures but just $t+1$ including the sender's.

[9]We conjecture that the protocol without these extra messages also satisfies consistency but the protocol as written has the same asymptotic complexity so we leave it as future work to prove it.

```
$\Pi^{t,\epsilon}_{\mathsf{BC}}(m^*)$

1. At time 0:
   – Set $\mathsf{sent}[m] = \mathsf{detect}[m] = \mathsf{false}$ for all $m \in M$ and $v = \bot$.
   – If $P_i = P^*$: set $\sigma^* \leftarrow \mathsf{as.Sign}(\mathsf{sk}_i, m^*)$ and $\mathsf{sent}[m^*] = \mathsf{detect}[m^*] = \mathsf{true}$,
     then multicast $(\sigma^*, \Sigma = (), \vec{P} = (), m^*)$.
2. For $r = 1, \ldots, t + R$: At time $r \cdot \Delta$: for each $m \in M$:
   – If received messages $(\sigma, \Sigma_1, \vec{P}_1, m), \ldots, (\sigma, \Sigma_j, \vec{P}_j, m)$ for distinct
     $\vec{P}_1, \ldots, \vec{P}_j$ from distinct parties such that for $\vec{P} = \cup_{k \in [1,j]} \vec{P}_j$
     (a) For each $k \in [1, j]$, $\mathsf{as.Verify}(\mathcal{VK}(\vec{P}_k), \Sigma_k, m) = 1$;
     (b) $\mathsf{as.Verify}(\mathcal{VK}(P^*), \sigma, m) = 1$;
     (c) $|\vec{P}| \geq \min\{r - 1, t\}$; and
     (d) $\{P_i, P^*\} \not\subseteq \vec{P}$:
       • If $\mathsf{sent}[m] = \mathsf{false}$ and $|\{m' \in M : \mathsf{sent}[m'] = \mathsf{true}\}| < 2$:
         ∗ Set $\mathsf{sent}[m] = \mathsf{true}$.
         ∗ Compute $\mathsf{as.Sign}(\mathsf{sk}_i, m) = \sigma$.
         ∗ Compute $\mathsf{as.Combine}(\Sigma', VK', m) = \sigma'$, where $\Sigma' =$
           $(\Sigma_1, \ldots, \Sigma_j, \sigma)$ and $VK' = (\mathcal{VK}(\vec{P}_1), \ldots, \mathcal{VK}(\vec{P}_j), \mathsf{vk}_i)$.
         ∗ For all $k \in [1, n]$, send $(\sigma_j, \sigma', \vec{P} \cup \{P_i\}, m)$ to $P_k$ with probability
           $q/n$.
         ∗ If $\mathsf{detect}[m] = \mathsf{false}$: set $\mathsf{detect}[m] = \mathsf{true}$ and multicast
           $(\sigma, (), (), m)$.
   – If received message $(\sigma, \cdot, \cdot, m)$ such that $\mathsf{as.Verify}(\mathcal{VK}(P^*), \sigma, m) = 1$:
       • If $\mathsf{detect}[m] = \mathsf{false}$ and $|\{m' \in M : \mathsf{detect}[m'] = \mathsf{true}\}| < 2$:
         ∗ Set $\mathsf{detect}[m] = \mathsf{true}$ and multicast $(\sigma, (), (), m)$.
3. At time $(t + R + 1) \cdot \Delta$:
   – If $\mathsf{sent} = \mathsf{detect}$ and $|\{m' \in M : \mathsf{sent}[m'] = \mathsf{true}\}| = 1$: set $v = m'$. Output
     $v$ and terminate.
```

Fig. 5: Synchronous broadcast (BC) protocol with sender P^* for $t < (1 - \epsilon) \cdot n$ and $\epsilon \in (0, 1)$ from the perspective of party P_i.

Finally, in step 3 of the protocol, if the maps sent and detect are equal and there is only one value $m' \in M$ such that $\mathsf{sent}[m'] = \mathsf{true}$, then P_i outputs this value and terminates, otherwise it outputs $\bot$ and terminates. Note if there are two or more messages m such that some honest party sets $\mathsf{sent}[m] = \mathsf{true}$, then the corresponding honest parties will broadcast the signer's signature on each m which all honest parties will process and thus terminate with $|\mathsf{detect}| = 2$ and output $\bot$.

Communication Complexity. Each party gossips at most two messages of size $O(\lambda + n + \ell)$ and multicasts at most two messages of size $O(\lambda + \ell)$. Since below we assume that $q = \Theta(\lambda)$, each gossip step requires sending an expected $O(\lambda)$ messages. Thus, communication complexity is overall $O(\lambda n^2 + n\lambda^2 + \ell(\lambda + n^2))$ which, when $\ell = O(\lambda)$, is $O(\lambda n^2 + \lambda^2 n) = O(\lambda n^2)$.

Theorem 3. *Let n, t be such that $t < (1-\epsilon) \cdot n$ for constant $\epsilon \in (0,1)$. Then $\Pi_{\mathsf{BC}}^{t,\epsilon}$ (Figure 5) is t-secure when run on a synchronous network and n-weakly valid when run on an asynchronous network.*

We prove this by proving the lemmas in the remainder of this subsection. Towards this goal, we begin by modeling probabilistic dissemination by the procedure AddRandomEdges (Figure 6) and prove some results used to prove t_s-security of our BC protocol. The techniques that follow are from [TLP22], where however they use binary input instead of multivalued one.

```
– Input: Set of n nodes $W$, disjoint subsets
  $S_2, S_3 \subset W$, $S \subset W \setminus (S_2 \cup S_3)$, integer $q \leq n$.
– Output: The graph $G$.
1. Let $G$ be the empty graph with node set $W$.
2. For every $u \in S$ and $v \in W$, add an edge $\{u, v\}$
   to $G$ with probability $q/n$.
3. Return graph $G$.
```

Fig. 6: $\mathsf{AddRandomEdges}(W, S_2, S_3, S, q)$ procedure.

AddRandomEdges is defined over a set of nodes W that is partitioned into three disjoint subsets $S_1, S_2, S_3 \subset W$ with $S_1 = W \setminus (S_2 \cup S_3)$. The way AddRandomEdges works is as follows. At the beginning, we have an empty graph G with node set W. Now given $S \subset S_1$, AddRandomEdges adds the edge $\{u, v\}$ to the graph G with probability q/n for every pair of nodes $u \in S$ and $v \in W$. At the end, the resulting graph with all the added edges is output. In the context of our protocol, S will be the set of parties that send a message m at a specific round r and S_2 will be the set of parties that have not received m in a previous round. An edge from $u \in S$ to $v \in W$ represents that party u sends m to party v in round r. Our goal is to determine how many parties in S_2 receive message m for the first time during round r. For this, we define the following indicator random variables.

Definition 17. *Let $G \leftarrow \mathsf{AddRandomEdges}(W, S_2, S_3, S, q)$. For all $u \in S_2$, let $Z_u \in \{0,1\}$ with $Z_u = 1$ if and only if u has nonzero degree in G.*

We find that the number of nodes in S_2 with nonzero degree in G is at least twice the number of nodes in S. This will allow us to show that messages propagate exponentially fast in our broadcast protocol. The proof of the following lemma can be found in the Appendix of [TLP22] (Proof of Lemma 1).

Lemma 5. *Let (S_1, S_2, S_3) be a partition of n nodes into (disjoint) sets with $\tau = |S_1| \leq \epsilon n/3$, $|S_2| = \epsilon n - |S_1|$, $|S_3| = n - \epsilon n$, where $\epsilon \in (0,1)$ is a constant.*

Let $S \subset S_1$ with $|S| \geq 2\tau/3$, and let $\{Z_u\}_{u \in S_2}$ be the random variables defined above. Then for $q \geq 15/\epsilon$, we have

$$\Pr\left[\sum_{u \in S_2} Z_u \geq 2\tau\right] = 1 - p, \quad \textit{where} \quad p = \max\left\{\epsilon n \cdot e^{-\epsilon q/9}, \left(\frac{e}{2}\right)^{-\epsilon q/4}\right\}.$$

For our proof of t_s-security and t_a-weak validity, we define the following sets of parties w.r.t. a value $m \in V$ and a round r:

1. $S(m, r)$: honest parties P_i that set $\mathsf{sent}[m] = \mathsf{true}$ at round r.
2. $S_1(m, r)$: honest parties P_i that set $\mathsf{sent}[m] = \mathsf{true}$ by round r.
3. $S_2(m, r)$: honest parties P_i that still have $\mathsf{sent}[m] = \mathsf{false}$ in round r.

Additionally, we let S_3 be the set of malicious parties (in particular, $|S_3| = n - \epsilon n$). Before we start we our proof of security for the Core BC Protocol, we show that the number of parties that receive a message at round $\tilde{r}$ that was sent at round $r < \tilde{r}$ increases exponentially with $\tilde{r} - r$ (with overwhelming probability). The proof of the following theorem can be found in the Appendix of [TLP22] (Proof of Lemma 2) with the difference that their set V is of order 2. However, this difference does not have any impact on the proof itself and therefore the same proof applies for our case of V being of order ≥ 2.

Lemma 6. *For a specific value $m \in V$, let r be the first round of the Core BC Protocol $\Pi_{\mathsf{BC}}^{t,\epsilon}$ where an honest party P_i sets $\mathsf{sent}[m] = \mathsf{true}$. Let $R = \lceil \log_3(\epsilon n) \rceil$, and let p be as in Lemma 5. Then we have the following bounds:*

1. *For all rounds ρ such that $r \leq \rho \leq r + R$ and $|S_1(m, \rho - 1)| \leq \epsilon n/3$, we have with probability at least $(1-p)^{\rho - r}$ that*

$$|S(m, \rho)| \geq 2/3 \cdot |S_1(m, \rho)| \quad \textit{and} \quad |S_1(m, \rho)| \geq 3^{\rho - r}.$$

2. *Let $\tilde{r} > r$ be a round such that $|S_1(m, \tilde{r} - 1)| > \epsilon n/3$. Then $|S_1(m, \tilde{r})| = \epsilon n$ with probability at least $(1 - \tilde{p})(1-p)^{\tilde{r} - r - 1}$ where $\tilde{p} = \epsilon n \cdot e^{-2\epsilon q/9}$.*

For the proofs hereafter, we let $R = \lceil \log_3(\epsilon n) \rceil$ and $q = \Theta(\lambda)$, where λ is the security parameter. Furthermore, all our statements hold with probability $1 - \mathsf{negl}(\lambda)$.

Lemma 7. *Let $t_s < (1 - \epsilon) \cdot n$. Then $\Pi_{\mathsf{BC}}^{t_s,\epsilon}$ achieves t_s-consistency when run in a synchronous network.*

Proof. Let $\mathcal{M} = \{m_1, \ldots, m_k\}$ be the set of messages for which at least one honest party sets $\mathsf{sent}[m_i] \leftarrow \mathsf{true}$ during one run of the protocol. We consider three cases separately, namely when $|\mathcal{M}| = 0$, $|\mathcal{M}| = 1$ and $|\mathcal{M}| > 1$. Suppose first that $|\mathcal{M}| = 0$. Then all honest parties will never update v and consequently output $\perp$ at step 3.

Suppose $|\mathcal{M}| = 1$. We show that if an honest party P_i sets $\mathsf{sent}[m] = \mathsf{true}$ for some value $m \in V$ at some round r, then by the end of the protocol all honest parties have set $\mathsf{sent}[m] = \mathsf{true}$ with probability $1 - \mathsf{negl}(\lambda)$. We consider the following two cases.

(i) Suppose $r < t_s + 1$. For this, we distinguish the two cases $S_1(m, r) > \epsilon n/3$ and $S_1(m, r) \leq \epsilon n/3$. If $S_1(m, r) > \epsilon n/3$, then by item 2 of Lemma 6 all ϵn honest parties set $\mathsf{sent}[m] = \mathsf{true}$ by the next round with probability at least $(1-\tilde{p})(1-p)^{(r+1)-r-1} = 1 - \epsilon n \cdot e^{-2\epsilon q/9}$. Since $n = \mathsf{poly}(\lambda)$, this probability is $1 - \mathsf{negl}(\lambda)$. On the other hand, if $S_1(m, r) \leq \epsilon n/3$, let $r_0 := r + R - 1$. In case $S_1(m, r_0) > \epsilon n/3$, again the previous case applies. Otherwise, item 1 of Lemma 6 tells us that at round $r_0 + 1 = r + R$ we get $S_1(m, r + R) \geq 3^R = 3^{\lceil \log_3(\epsilon n) \rceil} \geq 3^{\log_3(\epsilon n)} = \epsilon n$ with probability at least $(1-p)^R$. Since $-p \geq -1$ and $R \geq 1$, Bernoulli's inequality applies and gives $(1-p)^R \geq 1 - pR$. Since $n = \mathsf{poly}(\lambda)$ and $q = \Theta(\lambda)$, this probability is $1 - \mathsf{negl}(\lambda)$.

(ii) Suppose $r \geq t_s + 1$. Suppose an honest party P_i sets $\mathsf{sent}[m] = \mathsf{true}$ at some round $r \geq t_s + 1$. Then, P_i has received a valid multisignature on m of degree at least $t_s + 1$ and $|\{m' \in M : \mathsf{sent}[m'] = \mathsf{true}\}| \leq 1$. In particular, an honest party P_j already set $\mathsf{sent}[m] = \mathsf{true}$ at some round $r' < t_s + 1$. Hence, former case (i) applies to honest party P_j. Ultimately, all honest parties set $\mathsf{sent}[m] = \mathsf{true}$ by the end of the protocol with probability $1 - \mathsf{negl}(\lambda)$.

Finally, suppose that $|\mathcal{M}| \geq 2$. By the above logic, $\mathsf{sent}[m]$ was set to true for an honest party P_i in round $r < t_s + 1$ for each $m \in M$. By construction of Π_{BC}, P_i will set $\mathsf{detect}[m] \leftarrow \mathsf{true}$ and multicast $(\sigma, (), (), m)$ if not already done, where σ is the signature of the designated sender P^* on message m. Thus, in the next round (given $R \geq 1$), all honest parties will receive $(\sigma, (), (), m)$, which correctly verifies. If $|\mathcal{M}| = 2$, then all honest parties will set $\mathsf{detect}[m] = \mathsf{true}$ for both messages, and consequently at step 3 all output $\perp$ by construction of the protocol. If $|\mathcal{M}| > 2$, then it follows that for the first two messages that each honest party receives, they will set $\mathsf{detect}[m] \leftarrow \mathsf{true}$, and similarly output $\perp$. $\square$

Lemma 8. *Let $t_s < (1-\epsilon) \cdot n$. Then $\Pi_{\mathsf{BC}}^{t_s,\epsilon}$ achieves t_s-liveness and t_s-termination when run in a synchronous network.*

Proof. This follows trivially from the fact that all parties will terminate at step 3 after some finite amount of time regardless of whether or not they change the value v they output. $\square$

Lemma 9. *Let $t_s < (1-\epsilon) \cdot n$. Then $\Pi_{\mathsf{BC}}^{t_s,\epsilon}$ achieves t_s-validity when run in a synchronous network.*

Proof. In order to prove t_s-validity, we show that if the sender P^* is honest and inputs $m \in V$, then all honest parties output $v = m$. This directly follows from the proof of t_s-consistency and the fact that no honest party sets $\mathsf{sent}[m] \leftarrow \mathsf{true}$ on a message not signed by P^*. After $R = \lceil \log_3(\epsilon n) \rceil$ rounds, all honest parties have set $\mathsf{sent}[m] = \mathsf{true}$ with probability $1 - \mathsf{negl}(\lambda)$ and will output $v = m$. $\square$

Lemma 10. *Let $t_a \leq t_s$ and $t_a + 2 \cdot t_s < n$. Then $\Pi_{\mathsf{BC}}^{t_s,\epsilon}$ achieves t_a-weak validity even when run in an asynchronous network.*[10]

[10]Note that n-weak validity trivially follows.

Proof. In order to prove t_a-weak validity, we show that if the sender P^* is honest and inputs $m = m_j \in V$, then all honest parties output either $v = m_j$ or $v = \perp$. This directly follows from the proof of t_s-validity in the synchronous case: in case an honest party does not receive a valid multisignature on m_j by the end of the protocol run, it just outputs $v = \perp$. $\square$

B.2 Broadcast Extension Protocol for Dishonest Majority

We construct our broadcast extension protocol $\Pi^{t,\epsilon}_{\mathsf{BC\text{-}Ext}}$ which, for $t < (1-\epsilon) \cdot n$, allows for broadcast with $O(n\ell/\epsilon + \lambda n^2)$ communication complexity. This implies a $O(n\ell + \lambda n^2)$ honest majority broadcast algorithm (by setting $\epsilon = \frac{1}{2}$). Let $\Pi^{t,\epsilon}_{\mathsf{BC}}$ be a broadcast protocol (e.g. the one in the previous subsection) that terminates after T time with default value $\perp_{bc}$. We present the core protocol in Figure 7 which makes use of several helper functions defined in Figure 8.

Our protocol is very close to the dishonest majority broadcast protocol of Nayak et al. [NRS+20] (Figure 3). The only notable difference (modulo presentation differences) is that we gossip signatures at step 2(a) instead of multicasting.

$\Pi^{t,\epsilon}_{\mathsf{BC\text{-}Ext}}$ proceeds as follows. Consider the designated sender P^* with input message m^* Supposing P^* is honest, P^* splits up m^* into n blocks $D = \{D_1, \ldots D_n\}$ using the erasure coding scheme rs, where block D_j corresponds to party P_j. P^* then accumulates D into z using acc except that each value D_j is accumulated as (j, D_j). P^* then invokes $\Pi^{t,\epsilon}_{\mathsf{BC}}$ with input z. Hereafter, we assume that a value $z \neq \perp$ is then output by each honest P_i; if $z = \perp$, all honest parties wait and eventually output $\perp$.

As in Π_{BC} in the previous subsection, the protocol proceeds in $t + R$ rounds. Consider a value $z \neq \perp$ output by Π_{BC}. Supposing that P^* is honest, once Π_{BC} outputs, P^* first multicasts a signature on message HAPPY (after setting their own variable happy to true). Then, P^* computes witnesses w_j corresponding to block D_j for $j \in [1, n]$, and sends each party their respective block and the witness w_j (step 2(a)). On receipt of their witness (which they can verify is valid), P_i multicasts their block and share (step 2(b)). Note that P_i multicasts a block at most once. P_i then waits Δ time and collects blocks from all parties. On receipt of the first message (wit, s_j, w_j) from P_j, P_i can determine whether it is valid (and was accumulated in z) via $\mathsf{acc.Verify}(ak, z, w_j, (j, s_j))$. Finally, after Δ time, if P_i has received enough valid blocks, P_i can reconstruct a message m (step 2(c)). Then, by splitting up m and comparing the output of $\mathsf{acc.Eval}$ with z (output by Π_{BC}), P_i can determine whether they reconstructed a message that all other parties are able to or not. If so, they set variable $\mathsf{happy} = 1$.

Suppose an honest party sets $\mathsf{happy} = 1$. Then, that party splits up the message it learnt (or input to $\Pi_{\mathsf{BC\text{-}Ext}}$, if it is the sender P^*) into blocks, propagates signatures via gossip, and propagates blocks and their corresponding witnesses to each party individually. We show that all parties will eventually output the message, even with the use of gossip in step 2(a).

```
                    $\Pi^{t,\epsilon}_{\mathsf{BC\text{-}Ext}}(m^*)$

1. At time 0:
   – Set $o = \bot$ and $\mathsf{happy} = 0$.
   – Set $\mathsf{dist} = \mathsf{share} = \mathsf{recon} = \mathsf{false}$ and $\Sigma = VK = \vec{P} = \vec{D} = \bot$.
   – If $P_i = P^*$: set $o = m^*$ and $\mathsf{happy} = 1$. Compute $\vec{D} = \mathsf{EncodeBlocks}(m^*, n-t) = (D_1, \dots, D_n)$ and $z^* = \mathsf{acc.Eval}(ak, \vec{D})$. Invoke $\Pi^{t,\epsilon}_{\mathsf{BC}}$ with sender $P^*$ using input $z^*$.
2. At time $T$: If $z \neq \bot_{bc}$ is output by $\Pi^{t,\epsilon}_{\mathsf{BC}}$: for $r = 1, \dots, t+R$:
   (a) At time $T + (2r-2) \cdot \Delta$:
       – If $\mathsf{happy} = 1$ and $\mathsf{dist} = \mathsf{false}$:
         • Set $\mathsf{dist} = \mathsf{true}$.
         • If $P_i = P^*$: multicast $(\mathsf{as.Sign}(\mathsf{sk}_i, \mathsf{HAPPY}), \{P_i\})$. Otherwise, compute $\sigma = \mathsf{MultiSign}(\Sigma, VK)$, then for all $j \in [1, n]$, send $(\sigma, \vec{P} \cup \{P_i\})$ to $P_j$ with probability $q/n$.
         • Compute $(w_1, \dots, w_n) = \mathsf{acc.CreateWits}(ak, z, \vec{D})$.
         • For all $j \in [1, n]$, send $(\mathsf{wit}, s_j, w_j)$ to $P_j$ (where the $j$-th element of $\vec{D}$ is $(j, s_j)$).
   (b) At time $T + (2r-1) \cdot \Delta$:
       – If $\mathsf{share} = \mathsf{false}$ and received $(\mathsf{wit}, s_i, w_i)$ such that $\mathsf{acc.Verify}(ak, z, w, (i, s_i)) = 1$:
         • Set $\mathsf{share} = \mathsf{true}$ and multicast $(\mathsf{wit}, s_i, w_i)$.
   (c) At time $T + 2r \cdot \Delta$:
       – If $\mathsf{recon} = \mathsf{false}$:
         • For $j \in [1, n]$, let $(\mathsf{wit}, s_j, w_j)$ be the first message $(\mathsf{wit}, \cdot, \cdot)$ received from $P_j$ (set to $(\mathsf{wit}, \bot, \bot)$ if not received).
         • Set $S = (((1, s_1), w_1), \dots, ((n, s_n), w_n))$.
         • Compute $M = \mathsf{Reconstruct}(S, ak, z, t)$ and $\vec{D} = \mathsf{EncodeBlocks}(M, n-t)$.
         • If $\mathsf{acc.Eval}(ak, \vec{D}) = z$, $\mathsf{recon} = \mathsf{false}$, $\mathsf{HappyCheck}(r) = \mathsf{true}$ and $\mathsf{Valid}(M)$ ($\mathsf{HappyCheck}$ checks $r$ signatures were received):
           ∗ Set $\mathsf{happy} = 1$, $o = M$ and $\mathsf{recon} = \mathsf{true}$.
           ∗ Invoke $\mathsf{SigCombine}(r)$.
3. At time $T + 2 \cdot (t+R) \cdot \Delta$: Output $o$ and terminate.
```

Fig. 7: BC extension protocol with sender P^* for $t < (1-\epsilon) \cdot n$ and $\epsilon \in (0, 1)$ from the perspective of party P_i with respect to external validity predicate Valid.

Communication Complexity. Each party participates in Π_{BC} where the sender's input is of size $O(\lambda)$[11], which, using $\Pi^{t,\epsilon}_{\mathsf{BC}}$ from Figure 5, requires $O(\lambda n^2)$ communication. Each party sends at most one wit message to each party at step 2(b), which costs $O(n(\ell/b + \lambda) = O(\ell/\epsilon + \lambda n)$ for each party (recall $b = n - t$). The sender multicasts an $O(\lambda)$-sized message, and all parties gossip at most one

[11]Note that, to tame the communication complexity, that parties should disregard messages signed by a (dishonest) sender in Π_{BC} that are larger than the expected size of $O(\lambda)$.

Helper functions for $\Pi^{t,\epsilon}_{\mathsf{BC\text{-}Ext}}$

- $\underline{\mathsf{MultiSign}(\Sigma, VK)}$: Compute $\mathsf{as.Sign}(\mathsf{sk}_i, \mathsf{HAPPY}) = \sigma_i$. Compute $\mathsf{as.Combine}(\Sigma', VK', \mathsf{HAPPY}) = \sigma'$, where $\Sigma' = (\Sigma, \sigma)$ and $VK' = (VK, \mathsf{vk}_i)$. Output σ'.
- $\underline{\mathsf{HappyCheck}(r)}$: Output true if and only if received messages $(\Sigma^1, \vec{P^1}), \ldots, (\Sigma^j, \vec{P^j})$ for distinct $\vec{P^1}, \ldots, \vec{P^j}$ from distinct parties such that for $\vec{P} = \cup_{k \in [1,j]} \vec{P^k}$
  1. For each $k \in [1, j]$, $\mathsf{as.Verify}(\mathcal{VK}(\vec{P^k}), \Sigma^k, \mathsf{HAPPY}) = 1$;
  2. $|\vec{P}| \geq \min\{r, t + 1\}$; and
  3. $P_i \notin \vec{P}$.
- $\underline{\mathsf{SigCombine}(r)}$: Consider the messages $(\Sigma_1, \vec{P_1}), \ldots, (\Sigma_j, \vec{P_j})$ received that lead to $\mathsf{HappyCheck}(r)$ outputting true.
  - Set $\Sigma = \mathsf{as.Combine}(\Sigma = (\Sigma_1, \ldots, \Sigma_j), (\mathcal{VK}(\vec{P_1}), \ldots, \mathcal{VK}(\vec{P_j})), \mathsf{HAPPY})$, $VK = \cup_{k \in [1,j]} \mathcal{VK}(\vec{P_k})$ and $\vec{P} = \cup_{k \in [1,j]} \vec{P_k}$.
- $\underline{\mathsf{EncodeBlocks}(m, b)}$: Divide m into b evenly sized blocks $m_1, \ldots, m_b$ of length $\lceil \frac{\ell a}{b} \rceil$ where $\ell = \log_2 m$ and $a = \min\{i : n \leq 2^i - 1\}$. Compute $(s_1, \ldots, s_n) = \mathsf{rs.Encode}(m_1, \ldots, m_b)$. Output $((1, s_1), \ldots, (n, s_n))$.
- $\underline{\mathsf{Reconstruct}((((1, s_1), w_1), \ldots, ((n, s_n), w_n)), ak, z, d_0)}$: For $j \in [1, n]$: If $\mathsf{acc.Verify}(ak, z, w_j, (j, s_j)) = 0$, set $s_j = \bot$. Compute $m_1, \ldots, m_b = \mathsf{rs.Decode}(s_1, \ldots, s_n)$ and $m = (m_1 \mid \cdots \mid m_b)$. Output m.

Fig. 8: BC extension protocol (Fig. 7) helper functions from the perspective of party P_i.

message of size $O(\lambda + n)$, which costs an expected $O(\lambda^2 + \lambda n)$ per party. Thus the protocol incurs $O(n\ell/\epsilon + \lambda n^2 + \lambda^2 n) = O(n\ell/\epsilon + \lambda n^2)$ communication.

Theorem 4. *Let n, t be such that $t < (1 - \epsilon) \cdot n$ for constant $\epsilon \in (0, 1)$. Then $\Pi^{t,\epsilon}_{\mathsf{BC\text{-}Ext}}$ (Figures 7 and 8) is t-secure when run on a synchronous network and n-weakly valid when run on an asynchronous network.*

We prove this by proving the following lemmas. For the remainder of this section, we let $R = \lceil \log_3(\epsilon n) \rceil$ and $q = \Theta(\lambda)$, where λ is the security parameter. Furthermore, all our statements hold with probability $1 - \mathsf{negl}(\lambda)$.

Lemma 11. *Let $t_s < (1 - \epsilon) \cdot n$. Then $\Pi^{t_s,\epsilon}_{\mathsf{BC\text{-}Ext}}$ achieves t_s-liveness when run in a synchronous network.*

Proof. In order to prove t_s-liveness, we show that every honest party outputs a value v' and terminates. But this is clear from the protocol $\Pi^{t_s,\epsilon}_{\mathsf{BC\text{-}Ext}}$, especially step 3. □

Lemma 12. *If an honest party P_i reaches item (i) of step 2 of $\Pi^{t_s,\epsilon}_{\mathsf{BC\text{-}Ext}}$ by setting* $\mathsf{happy} = 1$ *and* $\mathsf{dist} = \mathsf{false}$ *and invokes the instructions of this step with input message $m \in V$, then every honest party P_j outputs $\sigma_j = m$.*

Proof. By t_s-consistency of the Core BC Protocol $\Pi_{\mathsf{BC}}^{t_s,\epsilon}$, the output z_i of $\Pi_{\mathsf{BC}}^{t_s,\epsilon}$ is the same at every honest party. If an honest party executes item (i) of step 2 with message m, then by definition $z_i = \mathsf{acc.Eval}(ak, \mathsf{EncodeBlocks}(m, n-t))$. If another honest party P_j sets $\sigma_j = m'$ after initialization, then it has to satisfy $\mathsf{acc.Eval}(ak, \mathsf{EncodeBlocks}(m', n-t)) = z_j = z_i$. By the properties of the Reed-Solomon code, the same codewords correspond to the same message. So if $m \neq m'$, then $\vec{D} \neq \vec{D}'$. In particular, there exists a component $D_i = (i, s_i) \in \vec{D}$ such that $D_i \notin \vec{D}'$. But a witness for $D_i \notin D'$ with respect to the accumulation value $z_i = \mathsf{acc.Eval}(ak, \vec{D}) = \mathsf{acc.Eval}(ak, \vec{D}')$ exists, which happens only with probability $\mathsf{negl}(\lambda)$ by security of the cryptographic accumulator. Therefore, we may assume $m = m'$ and only need to show that every other honest party P_j sets σ_j to a value.

Suppose that P_i executes item (i) of step 2 in some round r. In that case, P_i computes witnesses $(w_1, \ldots, w_n) = \mathsf{acc.CreateWits}(ak, z, \vec{D})$ and sends (wit, s_j, w_j) to each party P_j (where the j-th element of $\vec{D}$ is (j, s_j)). In item (ii) of step 2 of the same round r, every honest party can verify and multicast the valid (wit, s_j, w_j) to all the other parties if it has not done that already in a previous round. Note that verification of the tuples (wit, s_j, w_j) can be done safely because of the security of the accumulator. In item (iii) of step 2, every honest party gets at least $n - t_s$ correct coded values, since there are at least $n - t_s$ honest parties. In particular, every party P_j can identify the corrupted values and remove them, of which there are at most t_s. By the properties of the Reed-Solomon code, P_j with $\mathsf{happy} = 0$ is able to recover the message m. Now, the exact same analysis as in the proof of t_s-consistency of the Core BC Protocol $\Pi_{\mathsf{BC}}^{t,\epsilon}$ shows that with probability $1 - \mathsf{negl}(\lambda)$ after sufficiently many rounds in the domain $[1, t + R]$, every honest parties P_j has set its value $\mathsf{happy} = 1$ and $\sigma_j = m$. Note that an honest party does not set its output again in future rounds, since the multisignature already contains its signature. Once P_j sets σ_j, it will skip item (ii) of step 2 in all future round and the value of σ_j will not be changed. Therefore, in step 3 all honest parties output m and terminate. $\square$

Lemma 13. *Let $t_s < (1-\epsilon) \cdot n$. Then $\Pi_{\mathsf{BC\text{-}Ext}}^{t_s,\epsilon}$ achieves t_s-validity when run in a synchronous network.*

Proof. In order to prove t_s-validity, we show that if the sender P_j is honest and inputs $m = m_j \in V$, then all honest parties output $v = m_j$. In round $r = 1$, the sender executes item (i) of step 2 with message m_j. By the previous lemma, every honest party P_i outputs $\sigma_i = m_j$ by the end of the protocol and terminates. $\square$

Lemma 14. *Let $t_s < (1-\epsilon) \cdot n$. Then $\Pi_{\mathsf{BC\text{-}Ext}}^{t_s,\epsilon}$ achieves t_s-consistency when run in a synchronous network.*

Proof. In order to prove t_s-consistency, we show that every honest party outputs the same value σ'. If all honest parties output $\perp$, then they trivially output the same value $\sigma' = \perp$. Therefore, assume some honest party P_i outputs $\sigma_i = m \neq \perp$. In case P_i is the sender, t_s-validity of $\Pi_{\mathsf{BC\text{-}Ext}}^{t_s,\epsilon}$ tells us that all honest parties output m. So assume that P_i is not the sender. If P_i sets $\sigma_i = m \neq \perp$ in item

(iii) of step 2 of round $1 \leq r \leq t_s$, then P_i will execute the distribution item (i) of step 2 with message m in the next round $r+1$. By Lemma 12, all honest parties output the same value $\sigma' = m$. On the other hand, if P_i sets $\sigma_i = m \neq \bot$ in round $r \geq t_s + 1$, then it receives a multisignature of degree at least $t_s + 1$. In particular, one of these signatures comes from an honest party $P_j \neq P_i$ that has sent its signature (with probability q/n) and executed item (i) of step 2 in some previous round $1 \leq r' \leq t_s$. Again by Lemma 12, all honest parties (including P_i) output m' and therefore $m' = m$. So every honest party outputs the same value $\sigma' = m$. □

Lemma 15. *Let $t_a \leq t_s$ and $t_a + 2 \cdot t_s < n$. Then $\Pi^{t_s,\epsilon}_{\mathsf{BC\text{-}Ext}}$ achieves t_a-weak validity even when run in an asynchronous network.*

Proof. In order to prove t_a-weak validity, we show that if the sender P_j is honest and inputs $m = m_j \in V$, then all honest parties output either $v = m_j$ or $v = \bot$. This is clear from the proof of t_s-validity in the synchronous case and the construction of the protocol $\Pi^{t_s,\epsilon}_{\mathsf{BC\text{-}Ext}}$. □

C Asynchronous Agreement with Intrusion Tolerance

In this section, we modify the (multivalued) intrusion-tolerant Byzantine agreement protocol of Mostéfaoui and Raynal [MR17] that is t-secure (in asynchrony) given up to $t < n/3$ corruptions. Namely, we construct a protocol which is t_a-secure and t_s-valid with termination for t_a, t_s such that $0 \leq t_a < n/3 \leq t_s < n/2$ and $t_a + 2 \cdot t_s < n$. In the process, we extract a multivalued graded consensus protocol which may be of independent interest.

We introduce two primitives that we use to build Byzantine agreement, both of which are weaker primitives and thus do not require additional assumptions in asynchrony to solve (i.e. coin flipping). Looking forward, neither primitive guarantees termination itself but will terminate when used in our Byzantine agreement protocol. We note that for our MV-broadcast and graded consensus protocols in this section we assume that at most one message per 'type' is accepted by a party (which is trivial to implement). This step is taken in order to ensure security of the protocols and bound their communication complexity.

C.1 Towards Intrusion Tolerance: MV-Broadcast

The first primitive is MV-broadcast which was defined in [MR17] for the asynchronous setting. The goal of MV-broadcast is to 'filter' messages for consensus: parties input a value v and output a set S such that each value inside was either MV-broadcasted by an honest party (validity 1) or is a default value. It guarantees a limited form of agreement on values (validity 2 and inclusion). For MV-broadcast, we consider default value $\bot_{mv} \notin V$.

Definition 18 (Multivalued broadcast (MV-broadcast) [MR17]). *Let Π be a protocol executed by parties $P_1, \ldots, P_n$, where each party P_i begins holding input $v_i \in V$.*

– **Validity 1:** *Π achieves t-*validity 1 *if whenever at most t parties are corrupted, if an honest party outputs a set S such that $v \in S$ and $v \neq \bot_{mv}$, then v was input by an honest party.*
– **Validity 2:** *Π achieves t-*validity 2 *if the following holds whenever at most t parties are corrupted: if every honest party's input is equal to the same value v, then no honest party outputs a set containing $\bot_{mv}$.*
– **Inclusion:** *Π achieves t-*inclusion *if the following holds whenever at most t parties are corrupted: if honest P_i and P_j output sets S_i and S_j respectively, then $S_i = \{w\} \Rightarrow w \in S_j$ (note $w = \bot_{mv}$ is possible).*
– **Liveness:** *Π achieves t-*liveness *if whenever at most t parties are corrupted, every honest party outputs a set S where each $v \in S$ is such that $v \in V \cup \{\bot_{mv}\}$.*
– **Validity with liveness:** *Π achieves t-*validity with liveness *if the following holds whenever at most t parties are corrupted: if every honest party's input is equal to the same value v, every honest party outputs the set $S = \{v\}$.*

In [MR17], validity 1 and validity 2 are denoted as justification and obligation, respectively. We additionally define validity with liveness which our protocol will satisfy with t_s corruptions in synchrony. We present our MV-broadcast construction in Figure 9.

$\Pi_{\mathsf{MV}}^{t_a,t_s}(v_i)$

1. Set $\mathsf{output} = \mathsf{ready} = \mathsf{false}$ and $M_2(v) = \emptyset$ for all $v \in V$.
2. For every $v \in V$, let $M_1(v)$ be the (initially empty) set of parties from which P_i has received $(\mathsf{mv1}, v)$.
3. Multicast $(\mathsf{mv1}, v_i)$.
4. Upon receiving $(\mathsf{mv1}, v)$ messages on the same value v (possibly $\bot_{mv}$) from $n - t_s$ distinct parties: if $\mathsf{ready} = \mathsf{false}$, set $\mathsf{ready} = \mathsf{true}$ and multicast $(\mathsf{mv2}, v)$.
5. Upon receiving $(\mathsf{mv1}, v)$ messages on the same value v from $t_s + 1$ distinct parties: if $(\mathsf{mv1}, v)$ not yet multicast, multicast $(\mathsf{mv1}, v)$.
6. Upon receiving $(\mathsf{mv1}, v_{max})$ for v_{max} such that $v_{max} = \arg\max_{v' \in V} |M_1(v')|$: if $|\cup_{v' \in V} M_1(v')| - |M_1(v_{max})| \geq t_s + 1$ and $(\mathsf{mv1}, \bot_{mv})$ not yet multicast, multicast $(\mathsf{mv1}, \bot_{mv})$.
7. Upon having $|\cup_{v \in V} M_2(v)| \geq n - t_s$: if $\mathsf{output} = \mathsf{false}$, set $\mathsf{output} = \mathsf{true}$ and output $\{v : |M_2(v)| > 0\}$.

Updating $M_2(v)$:

(i) Upon receiving $(\mathsf{mv2}, v)$ from P_j: if previously received messages $(\mathsf{mv1}, v)$ on the same value from $n - t_s$ distinct parties, set $M_2(v) = M_2(v) \cup \{j\}$.
(ii) Upon receiving $(\mathsf{mv1}, v)$ messages on the same value v from $n - t_s$ distinct parties: set $M_2(v) = M_2(v) \cup \{j : P_j \text{ previously sent } (\mathsf{mv2}, v)\}$.

Fig. 9: MV-broadcast from the perspective of party P_i.

$\Pi_{\mathsf{MV}}^{t_a,t_s}$ works as follows. Consider honest P_i with input value $v_i \in V$. For each $v \in V$, P_i manages tracks two local initially empty sets $M_1(v)$ and $M_2(v)$. There are two types of messages multicast by P_i, $(\mathsf{mv1}, v)$ and $(\mathsf{mv2}, v)$ for $v \in V$. The distinction between $\mathsf{mv1}$ and $\mathsf{mv2}$ is simply to multicast $(\mathsf{mv2}, v)$ as soon as enough $(\mathsf{mv1}, v)$ messages (on the same value v) were received, which is tracked via the set $M_1(v)$. By definition, $M_1(v)$ is the set of parties from which P_i has received a $(\mathsf{mv1}, v)$ message. On the other hand, $M_2(v)$ keeps track of whether value v is validated (was input by an honest party) or not.

After variable initialisation, P_i multicasts $(\mathsf{mv1}, v_i)$ (step 3). For a given v, on first receipt of $(\mathsf{mv1}, v)$ from $t_s + 1$ parties, P_i multicasts it (step 5). On first receipt of $(\mathsf{mv1}, v)$ from $n - t_s$ parties for *any* v (step 4), P_i sets $\mathsf{ready} = \mathsf{true}$ and multicasts $(\mathsf{mv2}, v)$ i.e. P_i champions v for output. At this time, $M_2(v)$ is also updated (step (i)), If too many values were received, i.e. $|\cup_{v' \in V} M_1(v')| - |M_1(v_{max})| \geq t_s + 1$ where $v_{max} = \arg\max_{v' \in V} |M_1(v')|$, then P_i signals this by multicasting $(\mathsf{mv1}, \perp_{mv})$ (step 6). At any time, P_i updates the set $M_2(v)$ as $M_2(v) = M_2(v) \cup \{j\}$ after it receives proposal $(\mathsf{mv2}, v)$ by party P_j whenever P_i has received the message $(\mathsf{mv1}, v)$ from $\geq n - t_s$ distinct parties. Finally, when P_i first receives $n - t_s$ $(\mathsf{mv2}, w)$ messages where $n - t_s$ $(\mathsf{mv1}, w)$ messages were also received (on possibly different values w), captured by the predicate $|\cup_{v \in V} M_2(v)| \geq n - t_s$ in step 7, P_i outputs $\{v : |M_2(v)| > 0\}$.

Communication Complexity. We argue in the following that the communication complexity of our MV-broadcast protocol Π_{MV} is bounded by $O(n^3)$. Having a look at Figure 9, we see that every honest party multicasts at most one message of the form $(\mathsf{mv2}, -)$ (line 4). Furthermore, a message of the form $(\mathsf{mv1}, v)$ for some $v \neq \perp_{mv}$ is multicast if it was received from at least $t_s + 1$ distinct parties, ensuring that the message had to come from at least one honest party (line 5). In particular, an honest party only $\mathsf{mv1}$-multicasts at most $n - t_s$ times. Moreover, every honest party multicasts at most one message of the form $(\mathsf{mv1}, \perp_{mv})$ (line 6). As a result, the overall communication complexity of Π_{MV} is bounded by $O(n^3)$.

Theorem 5. *Let n, t_s, t_a be such that $0 \leq t_a < \frac{n}{3} \leq t_s < \frac{n}{2}$ and $t_a + 2 \cdot t_s < n$. Then MV-broadcast protocol $\Pi_{\mathsf{MV}}^{t_a,t_s}$ (Figure 9) satisfies t_s-validity 1, t_s-validity 2, t_s-valid with liveness, t_a-inclusion and t_a-liveness.*

We prove this by proving the following lemmas.

Lemma 16. *Let $t_s < n/2$. Then $\Pi_{\mathsf{MV}}^{t_a,t_s}$ achieves t_s-validity 1.*

Proof. Recall that a party P_i only outputs values v such that $|M_2(v)| > 0$ (line 7, Figure 9). In order to prove t_s-validity 1, we show that no honest party P_i adds a value $v \in V$ to $M_2(v)$ such that $(\mathsf{mv1}, v)$ was multicast only by dishonest parties. For this, let there be t_s dishonest parties that multicast $(\mathsf{mv1}, v)$ such that $(\mathsf{mv1}, v)$ is not multicast by any honest party. Since $t_s + 1 > t_s$, no honest party echoes the value v and thus v can be received only from the dishonest parties (line 5). As a consequence, no honest party P_i receives $(\mathsf{mv1}, v)$ from

$n - t_s > t_s$ distinct parties and so $M_2(v)$ is never populated by P_i (line 'Updating $M_2(v)$' (i) and (ii)). $\square$

Lemma 17. *Let $t_s < n/2$. Then $\Pi_{\mathsf{MV}}^{t_a,t_s}$ achieves t_s-validity with liveness.*

Proof. In order to prove t_s-validity with liveness, we show that if every honest party's input is equal to the same value $v \in V$, then every honest party outputs the set $S = \{v\}$.

Let every honest party multicast $(\mathsf{mv1}, v)$ on the same value $v \in V$ (line 3, Figure 9). As argued in the previous proof, no honest party echoes a value w different from v (line 5) Therefore, at most t_s values different from v are multicast as $(\mathsf{mv1}, -)$ messages. Now we consider an honest party P_i in the worst case execution, in which the t_s dishonest parties multicast $(\mathsf{mv1}, w)$ on the same value $w \neq v$. In that case, $|M_1(v)|$ increases monotonically from 0 to $n - t_s$ and $|M_1(w)|$ increases monotonically from 0 to t_s. There are the following two cases.

(i) Suppose that $|M_1(w)| \geq |M_1(v)|$. In this case, the predicate $|\bigcup_{v' \in V} M_1(v')| - |M_1(v_{max})| \geq t_s + 1$ in line 6 reduces to $|M_1(v)| \geq t_s + 1$ and returns false, since $|M_1(v)| \leq |M_1(w)| \leq t_s$. Hence, P_i never multicasts $(\mathsf{mv1}, \perp_{mv})$ (line 6).
(ii) Suppose that $|M_1(v)| \geq |M_1(w)|$. In this case, the predicate $|\bigcup_{v' \in V} M_1(v')| - |M_1(v_{max})| \geq t_s + 1$ in line 6 reduces to $|M_1(w)| \geq t_s + 1$ and returns false, since $|M_1(w)| \leq t_s$. Hence, P_i never multicasts $(\mathsf{mv1}, \perp_{mv})$ (line 6).

As a result, no honest party multicasts $(\mathsf{mv1}, \perp_{mv})$. Therefore, no honest party P_i receives $(\mathsf{mv1}, \perp_{mv})$ from $n - t_s > t_s$ distinct parties and $M_2(\perp_{mv})$ is never populated by P_i (line 'Updating $M_2(v)$' (i) and (ii)). The same is true for the value w. Consequently, every honest party outputs the same set $S = \{v\}$ (line 7). $\square$

Lemma 18. *Let $t_s < n/2$. Then $\Pi_{\mathsf{MV}}^{t_a,t_s}$ achieves t_s-validity 2.*

Proof. This follows directly from t_s-validity with liveness. $\square$

Lemma 19. *Let $t_a \leq t_s$ and $t_a + 2 \cdot t_s < n$. Then $\Pi_{\mathsf{MV}}^{t_a,t_s}$ achieves t_a-inclusion.*

Proof. In order to prove t_a-inclusion, we show that if honest parties P_i and P_j output sets S_i and S_j respectively, then $S_i = \{w\}$ implies $w \in S_j$ (note that $w = \perp_{mv}$ is possible).

Let honest party P_i output the set $S_i = \{w\}$ (line 7, Figure 9). This is conditioned on $|\cup_{v \in V} M_2(v)| \geq n - t_s$ by line 7 and implies that P_i has received $(\mathsf{mv2}, w)$ from at least $n - t_s$ distinct parties by definition of the set $M_2(-)$ in line 'Updating $M_2(v)$' (i) and (ii). Now consider an honest party $P_j \neq P_i$. Before P_j outputs the set S_j, it has received messages $(\mathsf{mv2}, -)$ from $n - t_s$ distinct parties, that is from at least $n - t_s - t_a > t_s$ honest parties. Since $(n - t_s) + (t_s + 1) > n$, it follows that there is an honest party P_k that sent the same message $(\mathsf{mv2}, v)$ to both P_i and P_j. But since P_i has received only messages $(\mathsf{mv2}, w)$ from $n - t_s$ parties, it follows that $v = w$ and thus $w \in S_j$. $\square$

Lemma 20. *Let $t_a \leq t_s$ and $t_a + 2 \cdot t_s < n$. Then $\Pi_{\mathsf{MV}}^{t_a,t_s}$ achieves t_a-liveness.*

Proof. In order to prove t_a-liveness, we show that every honest party outputs some set. For this, we first show that every honest party eventually multicasts some $(\mathsf{mv2}, -)$ message (line 4, Figure 9). We consider the following predicate P: There is a value $v \in V$ such that after some finite time at least $t_s + 1$ honest parties have multicast $(\mathsf{mv1}, v)$. Consider the following two cases.

(i) The predicate P is satisfied. In this case, every honest party eventually multicasts $(\mathsf{mv1}, v)$ by construction of $\Pi_{\mathsf{MV}}^{t_a,t_s}$ (line 5). Since there are at least $n - t_a \geq n - t_s$ honest parties, ready eventually is set to true by every honest party due to delivering $(\mathsf{mv1}, v)$ from $n - t_s$ distinct parties (line 4).
(ii) The predicate P is not satisfied. We consider an honest party P_i. Let $v_{max} = \max_{v' \in V} |M_1(v')$, i.e. the most frequent $\mathsf{mv1}$ value received, and let r be the number of honest parties that multicast $(\mathsf{mv1}, v_{max})$. Since P is not satisfied, $r \leq t_s$. Since there are at least $n - t_a$ honest parties, P_i receives at least $n - t_a + k$ messages $(\mathsf{mv1}, -)$ with some $k \in [0, t_a]$. At most $r + k$ of these parties sent message $(\mathsf{mv1}, v_{max})$ to P_i, and therefore at least $(n - t_a + k) - (r + k) = n - t_a - r$ of them sent values different from v_{max} to P_i. But since $n - t_a - r \geq n - t_a - t_s > t_s$, the predicate $|\bigcup_{v' \in V} M_1(v')| - |M_1(v_{max})| \geq t_s + 1$ in line 6 is satisfied and P_i multicasts $(\mathsf{mv1}, \bot_{mv})$ (line 6). Analogously, this applies to all honest parties. And thus, every honest party receives messages $(\mathsf{mv1}, \bot_{mv})$ from at least $n - t_a \geq n - t_s$ distinct parties. As a result, ready becomes eventually true for every honest party (line 4).

This concludes our initial assertion that every honest party eventually multicasts some message $(\mathsf{mv2}, -)$ (by setting ready to true) by line 4.

For the final step, we show that every honest party eventually receives *validated* messages $(\mathsf{mv2}, -)$ from at least $n - t_s$ distinct parties and thus outputs a set (namely $\{v : M_2(v) \neq \emptyset\}$) by line 7. Here, by a *validated* message $(\mathsf{mv2}, w)$ we mean one such that the party has also received messages $(\mathsf{mv1}, w)$ on the same value w from at least $n - t_s$ distinct parties (as is declared in line 4). By the previous assertion, each honest party P_i multicasts some message $(\mathsf{mv2}, v_i)$ where v_i is such that $|M_1(v_i)| \geq n - t_s$. Since $(n - t_s) - t_a > t_s$, at least $t_s + 1$ honest parties have multicast $(\mathsf{mv1}, v_i)$. Thus, every honest party eventually multicasts $(\mathsf{mv1}, v_i)$ and for every honest party P_j we have $|M_1(v_i)| \geq n - t_a \geq n - t_s$. As a consequence, every honest party P_j adds $\{j\}$ to its set $M_2(v_i)$. Since this proof hitherto also applies to every honest party P_j (in place of P_i), eventually every honest party receives validated messages $(\mathsf{mv2}, -)$ from at least $n - t_a \geq n - t_s$ distinct parties and thus outputs the set $\{v : M_2(v) \neq \emptyset\}$ (line 7). ◻

C.2 Validity-Optimized Graded Consensus

Next, we define a graded consensus primitive. In graded consensus, each party inputs a value but outputs both a value $v \in V \cup \{\bot\}$ and a corresponding grade $g \in \{0, 1, 2\}$. *Binary* graded consensus (i.e., where $V = \{0, 1\}$) was previously considered for building network-agnostic binary agreement in [BKL19].

Our primitive will also require an intrusion tolerance property that we define below.

Definition 19 (Graded consensus). *Let Π be a protocol executed by parties $P_1, \ldots, P_n$, where each party P_i begins holding input $v_i \in V$.*

- **Graded validity:** *Π achieves t-*graded validity *if the following holds whenever at most t parties are corrupted: if every honest party's input is equal to the same value v, then all honest parties output $(v, 2)$.*
- **Graded consistency:** *Π achieves t-*graded consistency *if the following holds whenever at most t parties are corrupted: (1) If two honest parties output grades g, g', then $|g - g'| \leq 1$. (2) If two honest parties output (v, g) and (v', g') with $g, g' \geq 1$, then $v = v'$.*
- **Liveness:** *Π achieves t-*liveness *if whenever at most t parties are corrupted, every honest party outputs (v, g) with either $v \in V$ and $g \geq 1$, or $v = \perp$ and $g = 0$.*
- **Intrusion tolerance:** *Π achieves t-*intrusion tolerance *if whenever at most t parties are corrupted, if (v, g) is output by an honest party and $g \geq 1$, then v was input by an honest party.*

We construct a multivalued graded consensus protocol $\Pi_{\mathsf{GC}}^{t_a,t_s}$ (Figure 10) Our protocol requires two (implicitly domain separated) instances of $\Pi_{\mathsf{MV}}^{t_a,t_s}$ which we denote by MV_1 and MV_2 with default values $\perp_{mv1}$ and $\perp_{mv2}$ respectively.

```
$\Pi_{\mathsf{GC}}^{t_a,t_s}(v_i)$

1. Set $\mathsf{output} = \mathsf{false}, \mathsf{val} = \perp, \mathsf{aux} = \perp$.
2. For every $v \in V$, let $R(v)$ be the set of parties from which $P_i$ has received
   $(\mathsf{msg}, v)$ for $\mathsf{msg} \in \{\mathsf{init}, \mathsf{echo}\}$. Initially, set $R(v) = \emptyset$ for all $v \in V$.
3. Multicast the message $(\mathsf{init}, v_i)$.
4. Upon receiving $(\mathsf{init}, v)$ messages on the same value $v$ from $t_s + 1$ distinct
   parties: if $v \neq v_i$ and $(\mathsf{echo}, v)$ not yet multicast, multicast $(\mathsf{echo}, v)$.
5. Upon receiving $(\mathsf{msg}, v)$ for some $\mathsf{msg} \in \{\mathsf{init}, \mathsf{echo}\}$ and $v$ when $\mathsf{output} = \mathsf{false}$:
   (a) If $v \neq v_i$ and $|R(v)| \geq t_s + 1$, set $\mathsf{output} = \mathsf{true}$ and $\mathsf{val} = \perp_{rd}$.
   (b) If $|R(v)| \geq n - t_s$, set $\mathsf{output} = \mathsf{true}$ and $\mathsf{val} = v$.
   (c) If $|\cup_{v' \in V} R(v')| - m \geq t_s + 1$ for $m = \max_{v' \in V} |R(v')|$, set $\mathsf{output} = \mathsf{true}$
       and $\mathsf{val} = \perp_{rd}$.
   (d) If $\mathsf{output} = \mathsf{true}$, run $\mathsf{MV}_1$ using input $\mathsf{val}$.
6. Upon receiving output $S_1$ from $\mathsf{MV}_1$: if $S_1 = \{v\}$, set $\mathsf{aux} = v$. Run $\mathsf{MV}_2$ using
   input $\mathsf{aux}$.
7. Upon receiving output $S_2$ from $\mathsf{MV}_2$:
   (a) If $S_2 = \{v\}$ and $v \notin \{\perp_{rd}, \perp_{mv1}, \perp_{mv2}, \perp\}$, output $(v, 2)$.
   (b) Otherwise, if $\exists\, v \in S_2$ s.t. $v \notin \{\perp_{rd}, \perp_{mv1}, \perp_{mv2}, \perp\}$, output $(v, 1)$.
   (c) Otherwise, output $(\perp, 0)$.
```

Fig. 10: Multivalued graded consensus from the perspective of party P_i.

$\Pi_{\mathsf{GC}}^{t_a,t_s}$ works as follows. Suppose (honest) P_i inputs v_i. In addition to messages from MV-broadcast, Π_{GC} uses two message types, namely (init, v) and (echo, v) for some $v \in V$. After initialising variables, P_i multicasts (init, v_i) (and doesn't send init messages hereafter). On receipt of messages (msg, v) (where $\mathsf{msg} \in \{\mathsf{init}, \mathsf{echo}\}$ from $t_s + 1$ parties (and thus an honest party input v), P_i multicasts (echo, v) if not yet done. Except for these steps, communication takes place through MV_1 and MV_2.

Let $R(v)$ be the set of parties from which P_i has received a message of the form (init, v) or (echo, v) (step 2). Then, whenever P_i receives a message (msg, v), P_i checks some conditions to determine whether it can input a value to MV_1 (after which the conditions are ignored). If $|R(v)| \geq t_s + 1$ and $v \neq v_i$, P_i inputs $\bot_{rd}$ to MV_1, indicating disagreement between two honest parties. Similarly, if $|\cup_{v' \in V} R(v')| - \max_{v' \in V} |R(v')| \geq t_s + 1$, P_i inputs $\bot_{rd}$ to MV_1. If $|R(v)| \geq n - t_s$, indicating enough parties received v, P_i inputs v to MV_1.

Then, P_i eventually outputs a set S_1 from MV_1. If $|S| = \{v\}$, P_i runs MV_2 with that input; otherwise P_i runs MV_2 on input $\bot$ (step 6). On outputting S_2 from MV_2 (step 2), if $S_2 = \{v\}$ and not a default value, then it outputs $(v, 2)$. Otherwise, if S_2 contains a non-default value, P_i outputs that value and $g = 1$; else, P_i outputs $(\bot, 0)$.

Communication Complexity. We argue in Lemma 25 that the communication complexity of our graded consensus protocol $\Pi_{\mathsf{GC}}^{t_a,t_s}$ is bounded by $O(n^3)$.

Theorem 6. *Let n, t_s, t_a be such that $0 \leq t_a < \frac{n}{3} \leq t_s < \frac{n}{2}$ and $t_a + 2 \cdot t_s < n$. Then graded consensus protocol $\Pi_{\mathsf{GC}}^{t_a,t_s}$ (Figure 10) satisfies t_s-graded validity, t_s-intrusion tolerance, t_a-graded consistency and t_a-liveness.*

We prove this by proving the following lemmas.

Lemma 21. *Let $t_s < n/2$. Then $\Pi_{\mathsf{GC}}^{t_a,t_s}$ achieves t_s-graded validity.*

Proof. In order to prove t_s-graded validity, we show that if every honest party's input is equal to the same value v, then all honest parties output $(v, 2)$. Let every honest party multicast (init, v) (line 3, Figure 10). Since there are at most $t_s < t_s + 1$ dishonest parties, no honest party echoes a value different from v (line 4). Therefore, from the perspective of every honest party P_i the set $R(w)$ is of size $\leq t_s$ for every $w \neq v$. In particular, conditions (a) and (c) of line 5 are not satisfied for the value v. On the other hand, since there are at least $n - t_s$ honest parties that multicast (init, v), it is $|R(v)| \geq n - t_s$, and P_i sets $\mathsf{output} = \mathsf{true}$ and $\mathsf{val} = v$ (line 5, condition (b)). As this applies to every honest party, each of them run MV_1 on the same input v (line 5 (d)). Now, t_s-validity with liveness from MV_1 ensures that every honest party outputs the set $S_1 = \{v\}$. As a result, each of them sets $\mathsf{aux} = v$ and runs MV_2 again on input v (line 6). The same argument yields that every honest party outputs the set $S_2 = \{v\}$. Finally, as $v \notin \{\bot_{rd}, \bot_{mv1}, \bot_{mv2}, \bot\}$, every honest party outputs $(v, 2)$ (line 7 (a)). $\square$

Lemma 22. *Let $t_a \leq t_s$ and $t_a + 2 \cdot t_s < n$. Then $\Pi_{\mathsf{GC}}^{t_a,t_s}$ achieves t_a-graded consistency.*

Proof. In order to prove t_a-graded consistency, we show that (1) if two honest parties output grades g, g', then $|g-g'| \leq 1$, and (2) if two honest parties output (v, g) and (v', g') with $g, g' \geq 1$, then $v = v'$.

For the first clause, assume that there are two honest parties P_i and P_j where P_i outputs $(v, 2)$. This means P_i's output S_2 from MV_2 is $S_2 = \{v\}$ with $v \notin \{\bot_{rd}, \bot_{mv1}, \bot_{mv2}, \bot\}$ (line 7 (a), Figure 10). By t_a-inclusion of MV_2, every honest party P_j outputs a set S_2 such that $v \in S_2$ and therefore can not output grade $g' = 0$ for its final decision (line 7 (c)). This proves the first clause.

For the second clause, consider two honest parties P_i and P_j that output (v, g) and (v', g') respectively with $g, g' \geq 1$. We write $S_2(P_k)$ for the output set S_2 from MV_2 of party P_k; for the output set S_1 we likewise define $S_1(P_k)$. In particular, $v \in S_2(P_i)$ and $v' \in S_2(P_j)$ with $v, v' \notin \{\bot_{rd}, \bot_{mv1}, \bot_{mv2}, \bot\}$ (line 7 (a) and (b)). By t_a-inclusion of MV_2, it follows that $v \in S_2(P_j)$ and thus $\{v, v'\} \subseteq S_2(P_j)$. By t_s-validity 1 of MV_2, v' was input by some honest party P_k to MV_2. In particular, $\mathsf{aux} = v'$ for party P_k and thus $S_1(P_k) = \{v'\}$ (line 6). Analogously, there is some honest party P_l that input v to MV_2 and hence $S_1(P_l) = \{v\}$. But by t_a-inclusion of MV_1, it follows that $v = v'$, which concludes the proof. □

Lemma 23. *Let $t_a \leq t_s$ and $t_a + 2 \cdot t_s < n$. Then $\Pi_{\mathsf{GC}}^{t_a,t_s}$ achieves t_a-liveness.*

Proof. In order to prove t_a-liveness, we show that every honest party outputs (v, g) with either $v \in V$ and $g \geq 1$, or $v = \bot$ and $g = 0$ (line 7, Figure 10). We consider the following predicate P: There is a value $v \in V$ such that after some finite time at least $t_s + 1$ honest parties have multicast (init, v). Consider the following two cases.

(i) The predicate P is satisfied. It follows that every honest party eventually multicasts (echo, v) (if not yet multicast (init, v)) (line 4). As there are at least $n - t_a \geq n - t_s$ honest parties, the predicate $|R(v)| \geq n - t_s$ in line 5 (b) becomes eventually true for every honest party and each of them sets $\mathsf{output} = \mathsf{true}$ and runs MV_1 (line 5 (d)). By t_a-liveness of MV_1, every honest party outputs some set S_1 and runs MV_2 on input aux (line 6). And by t_a-liveness of MV_2, every honest party outputs some set S_2 and finally outputs some graded value (v, g) (line 7). The requirement [$v \in V$ and $g \geq 1$, or $v = \bot$ and $g = 0$] is trivially satisfied by construction of $\Pi_{\mathsf{GC}}^{t_a,t_s}$ in its final step (line 7).

(ii) The predicate P is not satisfied. This means there is no value v that is multicast (init, v) by at least $t_s + 1$ honest parties. We consider an honest party P_i. Let v_{max} be its most often received value from distinct parties as defined in line 5 (c) and let r be the number of honest parties that multicast (init, v_{max}). By assumption, we have $r \leq t_s$. Let $k \in [0, t_a]$ be the number of distinct dishonest parties from which P_i has received the message (init, v_{max}). Clearly, at most $t_s + k \geq r + k$ distinct parties have sent the value v_{max} to P_i. Now assume that the waiting predicates [$|R(v)| \geq n - t_s$] and [for $w \neq v_i$ and $|R(w)| \geq t_s + 1$] in line 5 (a) and (b) are never satisfied (otherwise P_i sets $\mathsf{output} = \mathsf{true}$ and proceeds with the execution of MV_1). Since P_i does not terminate at these predicates, it receives a message from each honest party, that is from at least $n - t_a$ parties. Hence, P_i receives messages from at least $n - t_a + k$ distinct parties. As a consequence, at least $(n - t_a + k) - (t_s + k) = n - t_a - t_s > t_s$ distinct parties have sent values different from v to P_i. As a result, the predicate $| \cup_{v' \in V} R(v')| - |R(v_{max})| \geq t_s + 1$ in line 5 (c) is eventually satisfied and P_i runs MV_1 on input $\mathsf{val} = \bot_{rd}$. Therefore, every honest party runs MV_1 on some input (line 5 (d)) and the remainder of the proof proceeds as in the previous case.

□

Lemma 24. *Let $t_s < n/2$. Then $\Pi_{\mathsf{GC}}^{t_a,t_s}$ achieves t_s-intrusion tolerance.*

Proof. In order to prove t_s-intrusion tolerance, we show that no honest party P_i outputs a graded value (v, g) with $g \geq 1$ that was multicast (init, v) by dishonest parties only. For this, let there be t_s dishonest parties that multicast (init, v) such that (init, v) is not multicast by any honest party. Since $t_s + 1 > t_s$, no honest party echoes (echo, v) the value v and thus v can be received only from the t_s dishonest parties (line 4, Figure 10). As a consequence, for every honest party P_i it is $|R(v)| \leq t_s < n - t_s$ and thus no honest party sets val equal to v (line 5 (b)). The claim follows from t_s-validity 1 of MV_1 and MV_2. □

Finally, we bound the complexity of the protocol.

Lemma 25. *Let $t_s < n/2$. The total communication complexity of the graded consensus protocol $\Pi_{\mathsf{GC}}^{t_a,t_s}$ is bounded by $O(n^3)$.*

Proof. First, we show that each honest party may echo at most one message $(\mathsf{echo}, -)$. For this, let $n = 2t_s + 1$. Consider an honest party P_i. Clearly, P_i receives at most one message $(\mathsf{init}, -)$ from any other party, as it otherwise would know that the sender is dishonest. In order to multicast message (echo, v) for some $v \neq v_i$, P_i needs to receive (init, v) from $t_s + 1$ distinct parties (line 4, Figure 10). Since $n = (t_s + 1) + (t_s)$, it follows that P_i can echo at most one message $(\mathsf{echo}, -)$. This gives a communication complexity of $O(n^2)$ up to line 5 (c).

Next, we show that the communication complexity of MV_1 is bounded by $O(n^3)$. Having a look at Figure 9, we see that every honest party multicasts at most one message of the form $(\mathsf{mv2}, -)$ (line 4). Furthermore, a message of the form $(\mathsf{mv1}, v)$ for some $v \neq \bot_{mv}$ is multicast if it was received from at least $t_s + 1$ distinct parties, ensuring that the message had to come from at least one honest party (line 5). In particular, an honest party only $\mathsf{mv1}$-multicasts at most $n - t_s$ times. Moreover, every honest party multicasts at most one message of the form $(\mathsf{mv1}, \bot_{mv})$ (line 6). As a result, the overall communication complexity of MV_1 is bounded by $O(n^3)$. Getting back to Figure 10, line 5 (d) and line 6 give two instances of the MV-broadcast protocol with communication complexity $O(n^3)$, so that the overall communication complexity of $\Pi_{\mathsf{GC}}^{t_a,t_s}$ is bounded by $O(n^3)$.

□

C.3 Intrusion-Tolerant Multivalued Agreement

Let $\Pi_{\mathsf{BA}}^{t_a,t_s}$ be a Byzantine agreement protocol with input domain $\{0,1\}$ that is t_a-secure and t_s-valid with termination. We present a modified version of the protocol from [BKL19] in Appendix D with expected communication complexity $O(\lambda n^3)$ that satisfies these requirements. Let $\Pi_{\mathsf{GC}}^{t_a,t_s}$ be a multivalued graded consensus protocol that is t_a-secure and t_s-graded valid. We present intrusion-tolerant (multivalued) Byzantine agreement protocol $\Pi_{\mathsf{IT}}^{t_a,t_s}$ in Figure 11.

$\Pi_{\mathsf{IT}}^{t_a,t_s}(v_i)$

1. Set $\mathsf{aux}_i = \perp, \mathsf{bp} = 0$.
2. Run $\Pi_{\mathsf{GC}}^{t_a,t_s}$ using input v_i.
3. Upon receiving output (v, g) from $\Pi_{\mathsf{GC}}^{t_a,t_s}$: set $\mathsf{aux}_i = v$. If $g = 2$, set $\mathsf{bp} = 1$. Run $\Pi_{\mathsf{BA}}^{t_a,t_s}$ using input bp.
4. Upon receiving output b from $\Pi_{\mathsf{BA}}^{t_a,t_s}$: if $b = 1$, then multicast $\langle \mathsf{commit}, \mathsf{aux}_i \rangle_i$. Otherwise, multicast $\langle \mathsf{commit}, \perp \rangle_i$.
5. Upon receiving $t_s + 1$ signatures of $(\mathsf{commit}, \mathsf{aux})$ from distinct parties: collate signatures into Σ, multicast $(\mathsf{commit}, \mathsf{aux}, \Sigma)$, output aux and terminate.
6. Upon receiving $(\mathsf{commit}, \mathsf{aux}, \Sigma)$ where Σ consists of $t_s + 1$ valid signatures from distinct parties of $(\mathsf{commit}, \mathsf{aux})$: multicast $(\mathsf{commit}, \mathsf{aux}, \Sigma)$, output aux and terminate.

Fig. 11: Intrusion-tolerant multivalued Byzantine agreement from the perspective of party P_i.

We describe the protocol from the perspective of a party P_i with initial value $v_i \in V$. As usual, step 1 sets the helper variables $\mathsf{aux}_i, \mathsf{bp}$ to $\perp, 0$. In the first phase of the protocol, P_i runs the multivalued graded consensus protocol $\Pi_{\mathsf{GC}}^{t_a,t_s}$ on input v_i and gets the output (v, g) (step 2). The second phase then begins: P_i proposes 1 to $\Pi_{\mathsf{BA}}^{t_a,t_s}$ if $g = 2$ and otherwise proposes 0. If the output of $\Pi_{\mathsf{BA}}^{t_a,t_s}$ is 1, P_i signs and multicasts the value v (received previously from the output of the graded consensus protocol) along with the signature, otherwise it multicasts $\perp$ together with a signature (step 4). In the final phase of the protocol, P_i outputs (and terminates) a value aux if it received that value from or with at least $t_s + 1$ valid signatures, ensuring that at least one signature on aux is from an honest party (steps 5 and 6).

Communication Complexity. We argue in the following that the expected communication complexity of our intrusion-tolerant Byzantine agreement protocol $\Pi_{\mathsf{IT}}^{t_a,t_s}$ is $O(\lambda n^3)$. The graded consensus protocol $\Pi_{\mathsf{GC}}^{t_a,t_s}$ in line 2 has a communication complexity bounded by $O(n^3)$. The Byzantine agreement protocol $\Pi_{\mathsf{BA}}^{t_a,t_s}$ in line 3 has an expected communication complexity of $O(\lambda n^3)$. The multicast of commit messages in line 4 to 6 give an additional complexity of $O(n^2)$. As a result, the overall expected communication complexity of $\Pi_{\mathsf{IT}}^{t_a,t_s}$ is

$O(\lambda n^3)$. We note that we achieve this complexity by implementing the Byzantine agreement protocol $\Pi_{\mathsf{BA}}^{t_a,t_s}$ with a particular coin flip protocol from [GLL+21] (Algorithm 4).

Theorem 7. *Let n, t_s, t_a be such that $0 \le t_a < \frac{n}{3} \le t_s < \frac{n}{2}$ and $t_a + 2 \cdot t_s < n$. Then Byzantine agreement protocol $\Pi_{\mathsf{IT}}^{t_a,t_s}$ (Figure 11) is t_s-valid and t_a-secure with intrusion tolerance.*

We prove this by proving the following lemmas.

Lemma 26. *Let $t_a \le t_s$ and $t_a + 2 \cdot t_s < n$. Then $\Pi_{\mathsf{IT}}^{t_a,t_s}$ achieves t_s-validity with termination. It follows that $\Pi_{\mathsf{IT}}^{t_a,t_s}$ achieves t_a-validity.*

Proof. In order to prove t_s-validity with termination, we show that if every honest party's input is equal to the same value v, then every honest party outputs w and terminates. First, suppose that all honest parties receive output b from $\Pi_{\mathsf{BA}}^{t_a,t_s}$ and output v from $\Pi_{\mathsf{GC}}^{t_a,t_s}$ before terminating (line 3 and 4, Figure 11). Since all parties input v into $\Pi_{\mathsf{GC}}^{t_a,t_s}$, by t_s-graded validity of $\Pi_{\mathsf{GC}}^{t_a,t_s}$ all parties eventually receive output $(v, 2)$ from $\Pi_{\mathsf{GC}}^{t_a,t_s}$ (line 3). By construction of $\Pi_{\mathsf{IT}}^{t_a,t_s}$, all honest parties then propose $\mathsf{bp} = 1$ to $\Pi_{\mathsf{BA}}^{t_a,t_s}$ (line 3). By the t_s-validity of $\Pi_{\mathsf{BA}}^{t_a,t_s}$, all honest parties eventually output $b = 1$ from $\Pi_{\mathsf{BA}}^{t_a,t_s}$ (line 4). All $n - t_s$ honest parties P_i then multicast $\langle \mathsf{commit}, v \rangle_i$: since $n - t_s \ge t_s + 1$, all honest parties eventually deliver enough valid (commit, v) signatures or a valid $(\mathsf{commit}, v, \Sigma)$ message (line 5 and 6). Moreover, since $t_s + 1 > t_s$, no honest party receives a valid commit message for any other $v' \ne v$.

Suppose now that some honest P_i terminates before both outputting from $\Pi_{\mathsf{GC}}^{t_a,t_s}$ and outputting b from $\Pi_{\mathsf{BA}}^{t_a,t_s}$. That is, P_i delivered a valid $(\mathsf{commit}, v, \Sigma)$ or $t_s + 1$ valid $\langle \mathsf{commit}, v \rangle_i$ messages from distinct parties before multicasting $(\mathsf{commit}, v, \Sigma)$ to all honest parties (line 5 and 6), since by the t_s-validity of $\Pi_{\mathsf{BA}}^{t_a,t_s}$, bit $b = 0$ is never output by $\Pi_{\mathsf{BA}}^{t_a,t_s}$.[12] Similarly, honest parties only multicast their individual commit message upon $\Pi_{\mathsf{GC}}^{t_a,t_s}$ outputting a value (which must be $(v, 2)$ as argued above). It follows that all honest parties output the same v and terminate either on receipt of P_i's message $(\mathsf{commit}, v, \Sigma)$ or otherwise (line 5 and 6). □

Lemma 27. *Let $t_a \le t_s$ and $t_a + 2 \cdot t_s < n$. Then $\Pi_{\mathsf{IT}}^{t_a,t_s}$ achieves t_a-consistency.*

Proof. Suppose that all honest parties that terminate receive output from $\Pi_{\mathsf{BA}}^{t_a,t_s}$ and $\Pi_{\mathsf{GC}}^{t_a,t_s}$ before terminating; we argue similarly to above Lemma 26 if this is not the case. Suppose that some honest party outputs $(v, 2)$ from graded consensus (line 3). Then, by t_a-graded consistency, all honest parties output

[12] Note that this argument still holds even though not every party may actually input a value v to $\Pi_{\mathsf{BA}}^{t_a,t_s}$ (thus precluding t_s-validity). This is because these executions where some parties terminate before inputting v are indistinguishable from executions where these same parties input v and then halt for an indefinite period of time. Thus, the safety property contained in t_s-validity holds.

(v, g) with the same v where $g \in \{1, 2\}$, which all honest parties eventually do by t_a-liveness. Consider the first honest party who outputs b from $\Pi_{\mathsf{BA}}^{t_a,t_s}$ (line 4); by t_a-agreement and t_a-liveness all honest parties output the same b. Thus, all honest parties eventually multicast $\langle \mathsf{commit}, v \rangle_i$ if $b = 1$ or $\langle \mathsf{commit}, \perp \rangle_i$ if $b = 0$ (line 4). It follows that all honest parties who terminate output the same v by a similar argument to above.

Otherwise, if no party outputs $(v, 2)$, then by t_a-graded consistency no honest party proposes $\mathsf{bp} = 1$ to $\Pi_{\mathsf{BA}}^{t_a,t_s}$, and so by t_s-validity all parties output $b = 0$. Thus, as no honest party P_i multicasts $\langle \mathsf{commit}, v \rangle_i$ for $v \neq \perp$ and $t_s + 1 > t_a$ (line 4), all parties who terminate agree on output $\perp$. ☐

Lemma 28. *Let $t_a \leq t_s$ and $t_a + 2 \cdot t_s < n$. Then $\Pi_{\mathsf{IT}}^{t_a,t_s}$ achieves t_a-liveness and t_a-termination.*

Proof. Note that, in $\Pi_{\mathsf{IT}}^{t_a,t_s}$, t_a-liveness holds if and only if t_a-termination holds since all parties terminate directly after outputting.

Suppose that all honest parties that terminate receive output from $\Pi_{\mathsf{BA}}^{t_a,t_s}$ and $\Pi_{\mathsf{GC}}^{t_a,t_s}$ before terminating; we argue similarly to above Lemma 26 if this is not the case. By the t_a-liveness of $\Pi_{\mathsf{GC}}^{t_a,t_s}$, all honest parties that output eventually output (v, g) (line 3), and by t_a-agreement and t_a-liveness of $\Pi_{\mathsf{BA}}^{t_a,t_s}$, all honest parties output the same b (line 4). All honest parties then multicast the same signed (commit, v) pair (line 4), and then since $n - t_a > t_s + 1$ and similarly to before all honest parties eventually output a value and terminate (line 5 and 6). ☐

Lemma 29. *Let $t_a \leq t_s$ and $t_a + 2 \cdot t_s < n$. Then $\Pi_{\mathsf{IT}}^{t_a,t_s}$ achieves t_s-intrusion tolerance.*

Proof. This follows similarly from the proof of t_s-validity with termination, Lemma 26. Suppose that some honest party P_i outputs $(v, 2)$ from the graded consensus, i.e. $\Pi_{\mathsf{GC}}^{t_a,t_s}$ (line 3). By its t_s-intrusion tolerance, v must have been input to $\Pi_{\mathsf{GC}}^{t_a,t_s}$, and thus $\Pi_{\mathsf{IT}}^{t_a,t_s}$, by an honest party. Then by t_s-consistency of $\Pi_{\mathsf{GC}}^{t_a,t_s}$, all honest parties who output will output (v, g) with $g \geq 1$ (line 3). Thus, if 1 is output by all honest parties in $\Pi_{\mathsf{BA}}^{t_a,t_s}$ (by properties of $\Pi_{\mathsf{BA}}^{t_a,t_s}$), it follows that $\mathsf{aux}_i = v$ will be output by all honest parties; otherwise, $\perp$ will be output. Thus, t_s-intrusion tolerance holds in this case.

Suppose now that no honest party outputs (v, g) from $\Pi_{\mathsf{GC}}^{t_a,t_s}$ such that $g = 2$. By construction of $\Pi_{\mathsf{IT}}^{t_a,t_s}$, no honest party sets $\mathsf{bp} = 1$ (note all parties output from $\Pi_{\mathsf{GC}}^{t_a,t_s}$ by its t_s-liveness) and thus all honest parties propose $\mathsf{bp} = 0$ to $\Pi_{\mathsf{BA}}^{t_a,t_s}$. By t_s-validity of $\Pi_{\mathsf{BA}}^{t_a,t_s}$, it follows that all honest parties decide 0, and thus by construction of $\Pi_{IT}^{t_a,t_s}$ will eventually output $\perp$. That is, they will not output a value proposed by a dishonest party. ☐

D Binary Agreement Protocol

We present a Byzantine binary agreement protocol $\Pi_{\mathsf{BA}}^{t_a,t_s}$. Let $\Pi_{\mathsf{GC}}^{t_s}$ be a (binary) graded consensus protocol, i.e. takes as input a value in $\{0, 1\}$ where each party outputs a value $v \in \{0, 1, \bot\}$ and a grade $g \in \{0, 1, 2\}$. We require that $\Pi_{\mathsf{GC}}^{t_s}$ satisfies t_s-graded validity and is t_a-secure: the protocol from [BKL19] (Figure 4) satisfies these properties with $O(n^2)$ communication complexity. We also require a coin-flip mechanism $\mathsf{CoinFlip}$ which allows all parties to generate and know an unbiased binary value $\mathsf{Coin}_r \in \{0, 1\}$ for $r \geq 2$. Upon receiving input $r \geq 2$ from $t_s + 1$ parties, the coin flip mechanism generates an unbiased coin $\mathsf{Coin}_r \in \{0, 1\}$ and sends (r, Coin_r) to all parties. In particular, if at most t_s parties are corrupted, at least one honest party must send r to $\mathsf{CoinFlip}$ before the adversary can learn the coin Coin_r. We will rely on a $\tilde{p}$-weak coin flip with $\tilde{p} = 1/3$, where honest parties agree on the coin only with probability $\tilde{p} < 1$. This comes with an increase in the expected round complexity by a factor of $O(1/\tilde{p}) = O(1)$. Our coin flip mechanism is the one from [GLL+21] (Algorithm 4), which costs only $O(\lambda n^3)$ bits and has expected constant rounds (and ensures that with probability at least 1/3, all honest parties output an unbiased common coin).

Our asynchronous Byzantine agreement protocol $\Pi_{\mathsf{BA}}^{t_a,t_s}$ (Figure 12) works as follows. We describe it from the perspective of a party P_i with input value v_i. In the protocol, we say message $(\mathsf{commit}, b, \sigma)$ from party P_i (as in 'Termination procedure', steps (i) and (ii)) is valid if $b \in \{0, 1\}$ and σ is a valid signature from P_i on (commit, b), i.e $\sigma \leftarrow \langle \mathsf{commit}, b \rangle_i$. Also, we say that a set of signatures is a certificate for b (as in step (i) and (ii)) if the set contains valid signatures on (commit, b) from at least $t_s + 1$ distinct parties.

As usual, at the beginning helper variables are set (step 1). Afterwards, the protocol proceeds in round defined by the parameter r. In such a round, P_i first runs the graded consensus protocol $\Pi_{\mathsf{GC}}^{t_s}$ on input $b = v_i$ with (b, g) being the output (step 2). Note that for rounds $r < 3$ we set the coin to some deterministic default value (which does not affect the safety of the algorithm). We do this because we can can then assume our coin flip mechanism only works in asynchrony (and thus use the algorithm from [GLL+21]) so that when we show t_s-validity with termination, it will also hold that the coin is never used. Otherwise, $\mathsf{CoinFlip}(r)$ is invoked.

Then, if $g < 2$, P_i runs $\Pi_{\mathsf{GC}}^{t_s}$ on input Coin_r (step 4), otherwise on input b (steps 4 and 5), with (b', g) being the output. Now if $g > 0$, set $b = b'$ (step 5). In case $g = 2$, the next step is done only once (which is guaranteed by setting helper variable cm to true): compute the signature σ on (commit, b) and multicast the message $(\mathsf{commit}, b, \sigma)$ (step 7). Finally, the next round starts by increasing r by one (step 8) with a small restriction: As soon as P_i computes a commit message, it only executes one additional round of the protocol and then stops (this is ensures by the variable stop in steps 7 and 8). Furthermore, party P_i terminates and ends the protocol as soon as (i) it receives valid commit messages on the same value b from at least $t_s + 1$ distinct parties (before termination, P_i combines the signatures into a certificate Σ, multicasts $(\mathsf{notify}, b, \Sigma)$ and outputs b), or (ii) it receives $(\mathsf{notify}, b, \Sigma)$ with Σ being a certificate on messages (commit, b) for the same value b (before termination, P_i multicasts $(\mathsf{notify}, b, \Sigma)$ and outputs b).

The purpose of the stop variable is to ensure that the communication complexity of the protocol stays bounded. The protocol is well-defined and terminates, since we have the following: if one honest party set $\mathsf{cm} = \mathsf{true}$ in round r, then all parties set $\mathsf{cm} = \mathsf{true}$ in (at most) round $r + 1$. We sketch the argument for this claim in the following. We will say a party *commited* if it sets $\mathsf{cm} = \mathsf{true}$. Suppose an honest party commited in round r. By construction, it must have $g = 2$ and by graded consistency of $\Pi_{\mathsf{GC}}^{t_s}$, it is $|g - g'| \leq 1$ for all g' output by honest parties. Therefore, all honest parties output $g = 1$ or $g = 2$. Furthermore, graded consistency implies that if one honest party outputs $(v, g = 2)$, then all honest parties output the same v, and so all honest parties will input v in the next round. Again by graded validity, all honest parties will output $g = 2$ from the first execution of $\Pi_{\mathsf{GC}}^{t_s}$ and by similar arguments every honest party will have commited in that round (we will see this argumentation more thoroughly in the proof for t_s-graded validity below).

```
$\Pi_{\mathsf{BA}}^{t_a,t_s}(v_i)$

1. Set $\mathsf{cm} = \mathsf{output} = \mathsf{false}$, $\mathsf{stop} = 0$, $b = v_i$, and $r = 1$.
   While $\mathsf{output} = \mathsf{false}$ do:
2. Run $\Pi_{\mathsf{GC}}^{t_s}$ on input $b$. Let $(b, g)$ be the output.
3. If $r < 3$, set $\mathsf{Coin}_r = 1$. Otherwise, let $\mathsf{Coin}_r \leftarrow \mathsf{CoinFlip}(r)$.
4. If $g < 2$, set $b = \mathsf{Coin}_r$.
5. Run $\Pi_{\mathsf{GC}}^{t_s}$ on input $b$. Let $(b', g)$ be the output.
6. If $g > 0$, set $b = b'$.
7. If $g = 2$ and $\mathsf{cm} = \mathsf{false}$:
     – Set $\mathsf{cm} = \mathsf{true}$ and $\mathsf{stop} = r + 1$.
     – Compute $\sigma \leftarrow \langle \mathsf{commit}, b \rangle_i$.
     – Multicast $(\mathsf{commit}, b, \sigma)$.
8. If $\mathsf{stop} > r$, set $r = r + 1$ (and repeat from step 2).

Termination procedure:

(i) Upon receiving (valid) commit messages $(\mathsf{commit}, b, \sigma)$ on the same value $b$
    from $t_s + 1$ distinct parties:
     • Combine the signatures into a certificate $\Sigma = (\sigma_1, \ldots, \sigma_{t_s+1})$.
     • Multicast $(\mathsf{notify}, b, \Sigma)$.
     • Output $b$, set $\mathsf{output} = \mathsf{true}$, and terminate.
(ii) Upon receiving $(\mathsf{notify}, b, \Sigma)$ such that $\Sigma$ contains valid signatures on commit
    messages $(\mathsf{commit}, b)$ on the same value $b$ from $t_s + 1$ distinct parties:
     • Multicast $(\mathsf{notify}, b, \Sigma)$.
     • Output $b$, set $\mathsf{output} = \mathsf{true}$, and terminate.
```

Fig. 12: Binary agreement protocol from the perspective of party P_i.

We have the following theorem.

Theorem 8. *Let $t_a \leq t_s$ and $t_a + 2 \cdot t_s < n$. Then Π_{BA} achieves t_s-validity with termination and is t_a-secure.*

Proof. First, we prove the t_s-validity with termination. For this, assume all honest parties initially hold the same value $v \in \{0, 1\}$. All honest parties use v as input in the first execution of the graded consensus protocol $\Pi_{\mathsf{GC}}^{t_s}$ (step 2). By t_s-graded validity of $\Pi_{\mathsf{GC}}^{t_s}$, all honest parties output $(v, 2)$ from that execution. Thus, all honest parties run a second instance of $\Pi_{\mathsf{GC}}^{t_s}$ using input v (step 5), again receiving $(v, 2)$ as output. Therefore, all honest parties multicast a commit message on v (step 7). Additionally, by the stop variable, all honest parties will only execute one additional round of the protocol. Furthermore, the honest parties will receive at most $t_s < t_s + 1$ commit messages on some $w \neq v$. Therefore, all honest parties eventually receive valid commit messages on v from $n - t_s \geq t_s + 1$ distinct parties, output v, and terminate (step (i) and (ii) of 'Termination procedure').

The proof of the t_a-security follows the same argumentation as the proof of Lemma 11 in [BKL19] and therefore we skip it here.

E Deferred Security Proofs

Lemma 30. *Let $0 \leq t_a < n/3 \leq t_s < n/2$, $t_a + 2 \cdot t_s < n$ and $d = t_s + 1$. Then distributed key generation protocol $\Pi_{\mathsf{DKG}}^{t_a,t_s}$ (Figure 2) achieves (t_s, d)-correctness and t_s-consistency when run in a synchronous network and (t_a, d)-correctness and t_a-consistency when run in an asynchronous network.*

Proof. First, suppose that at most t_s parties are corrupted and that the network is synchronous.

Note that all parties terminate the n instances of $\Pi_{\mathsf{BC\text{-}Ext}}$ at time T (t_s-liveness) with the same values $(M'_1, \ldots, M'_n)$ (t_s-consistency) at step 2. By t_s-validity and t_s-external validity of $\Pi_{\mathsf{BC\text{-}Ext}}$ and since at least $n - t_s$ parties are honest, at least $n - t_s$ instances of $\Pi_{\mathsf{BC\text{-}Ext}}$ terminate with valid input from honest parties. Thus, all honest parties satisfy the condition at step 2(a) and invoke Split with the same input. Since $\mathsf{acc.Eval}$ and $\mathsf{acc.CreateWits}$ are deterministic, all honest parties invoke $\Pi_{\mathsf{IT}}^{t_a,t_s}$ with the same input z. Thus by t_s-validity of Π_{IT} all honest parties output z.

By n-external validity of $\Pi_{\mathsf{BC\text{-}Ext}}$, each value $M'_i \neq \bot_{bc}$ output by instance i is of the form $(C_i = (C_{i0}, \ldots, C_{it_s}), c_i = (c_{i1}, \ldots, c_{in}), \pi_i = (\pi_{i1}, \ldots, \pi_{in}))$. By the completeness and soundness of nizk_1, each c_{ij} is an encryption of $(s_{ij}, u_{ij}) = (f_i(j), f'_i(j))$ under ek_j for polynomials f_i, f'_i defined by the values in C_i in the exponent of g, h. Note that Split is defined such that each message L_j contains:

- The same set of qualified parties Q;
- c^*_j, i.e. encryptions of $f_q(j)$ under pk_j for $q \in Q$; and
- $C^*_j = \prod_{q \in Q} C^*_{j,q}$ where $C^*_{j,q}$ is $f_q(j)$ and $f'_q(j)$ evaluated in the exponent of g and h respectively. Thus, C^*_j is $\sum_{q \in Q} f'_q(j), \sum_{q \in Q} f'_q(j)$ evaluated in the exponent of g, h.

By construction of steps 4 and 5 and the security of acc, all honest parties eventually reach step 6 on receipt of their valid part message derived from the output of the n instances of $\Pi_{\mathsf{BC\text{-}Ext}}$ which all parties agree on. Each party then multicasts a recon message. Similarly to the above, by the security and correctness of acc and nizk_2, all honest parties eventually reach step 7. By construction, each honest party performs Lagrange interpolation in the exponent of g with respect to $t_s + 1$ valid shares with respect to the polynomial $F(\cdot) = \sum_{q \in Q} f_q(\cdot)$.

Now, t_s-consistency follows since all honest parties evaluate $F(\cdot)$ at 0 in the exponent to derive the same y and at $[1, n]$ to derive the same sequence sequence $(\mathsf{ps}_1, \ldots, \mathsf{ps}_n)$. (t_s, d)-correctness follows where the polynomial F in the definition of DKG (Definition 8) is as described above.

Suppose now that the network is asynchronous and at most t_a parties are corrupted. By n-external validity of $\Pi_{\mathsf{BC\text{-}Ext}}$, all non-bottom messages that honest parties output at the beginning of step 2 satisfy $\mathsf{Valid}()$, and by construction all parties then invoke Π_{IT} with some input.

- Suppose that honest party P_i outputs $v \in \{\bot_{it}, \bot_{dkg}\}$ from Π_{IT}. Then all honest parties eventually (t_a-consistency) output the same value v (t_a-validity). It follows that no honest party sets $\mathsf{ready} = \mathsf{true}$. Since all honest parties thus invoke Π_{ADKG}, (t_a, d)-correctness and t_a-consistency follows from the (t_a, d)-security of Π_{ADKG}.
- Otherwise, by t_a-intrusion tolerance (and t_a-security) of Π_{IT}, all parties eventually output the same value z_{it} which is such that z_{it} was proposed by an honest party, say P_i. Similarly to the synchronous case, P_i must have output at least $n - t_s$ well-formed values M'_i at step 2, and then at step 4 sent valid part messages to all parties. Thus, all honest parties eventually set $\mathsf{ready} = \mathsf{true}$. Since there are at least $t_s + 1$ honest parties and by similar arguments to the synchronous case, it follows that all parties eventually reach step 7, from which the two claimed properties similarly follow.

We note also that honest parties can terminate upon generating output since they do not send any new messages thereafter. □

Lemma 31. *Let $0 \le t_a < n/3 \le t_s < n/2$, $t_a + 2 \cdot t_s < n$. Then distributed key generation protocol $\Pi^{t_a,t_s}_{\mathsf{DKG}}$ (Figure 2) achieves t_s-secrecy and t_s-uniformity when run in a synchronous network and t_a-secrecy and t_a-uniformity when run on an asynchronous network.*

Proof. We first consider secrecy. We follow the same high-level strategy used in previous work [GJKR99,GJKR07,SBKN21]. We construct a simulator S which takes as input a public key y, and the set of initially corrupted parties by adversary $\mathcal{A}$ (recall we consider static corruptions), which w.l.o.g. we write as $B = \{P_1, \ldots P_t\}$ with $t \le t_s$. $\mathcal{A}$ controls the network and co-ordinates the actions of parties in B. To prove t_s-secrecy, we show that S can simulate interaction with $\mathcal{A}$ such that, conditioned on the output public key being y, the view of the run from $\mathcal{A}$'s perspective is indistinguishable from one where the specification of Π_{DKG} is exactly executed. We present the simulator S in Figure 13.

Simulator $S(y, B)$ for $\Pi_{\mathsf{DKG}}^{t_a,t_s}$

1. Simulate for G by locally executing steps 1 to 5 of Π_{DKG} (Figure 2) until *any* party $P_i \in G$ outputs a value $v \in \{\bot_{it}, \bot_{dkg}\}$ from $\Pi_{\mathsf{IT}}^{t_a,t_s}$. In this case, begin executing the simulator for Π_{ADKG} with input (y, B) (while continuing to simulate steps 1 to 5 of Π_{DKG}) and ignore steps 2 to 4 below; when Π_{ADKG} produces output o for $P_i \in G$, output o and terminate.
2. Upon the *first* party $P_i \in G$ reaching step 6 of Π_{DKG} with $\mathsf{ready} = \mathsf{true}$:
   (a) Let $P_j \in G$ be a party that executed step 4 of Figure 2 with $\mathsf{it} = \mathsf{bc} = \mathsf{true}$ and $z = z_{it}$. Consider the set of messages $\mathcal{M}$ of the form $(\mathsf{part}, L_k = (c_k^*, C_k^*, Q), w_k)$ that it sent to P_k for each $k \in [1, n]$.
   (b) For $q \in Q$, compute polynomials $f_q(y)$ from $\mathcal{M}$ by Lagrange interpolation of t_s+1 points (s_{qk}, k) obtained by decrypting t_s+1 ciphertexts c_{qk} using simulated secret keys dk_k (where $P_k \in G$).
   (c) For $k \in [1, t_s]$, compute $x_j = \sum_{j \in Q} f_j(k)$ using polynomials $f_j(y)$ derived above. Let $g^{F(y)}$ be the exponential w.r.t. g of the polynomial defined by Lagrange interpolation in the exponent of the $t_s + 1$ points $\{(0, y), (1, g^{x_1}), \ldots, (t_s, g^{t_s})\}$.
   (d) For $k \in [t+1, n]$, compute $D_k = g^{F(k)}$, and compute C_k'' as in step 6 of Figure 2. Run the nizk_1 simulator with respect to (D_k, C_k'') which outputs π_k'. Compute $M_k = (\mathsf{recon}, D_k, \pi_k', L_k, w_k)$ where (L_k, w_k) is derived from $(\mathsf{part}, L_k, w_k) \in \mathcal{M}$.
3. Upon party $P_i \in G$ reaching step 6 of Figure 2 with $\mathsf{ready} = \mathsf{true}$: instead of executing step 6, simulate P_i multicasting M_i for M_i defined above.
4. Upon party $P_i \in G$ reaching step 7 of Figure 2 simulate locally.

Fig. 13: Simulator for $\Pi_{\mathsf{DKG}}^{t_a,t_s}$ (Fig. 2) from the perspective of the simulator S interacting with adversary $\mathcal{A}$, where $B = \{P_1, \ldots, P_t\}$ is the set of parties that $\mathcal{A}$ initially corrupts, where $t \le t_s$ (resp. $t \le t_a$) in synchrony (resp. asynchrony), and $G = \{P_{t+1}, \ldots, P_n\}$ (without loss of generality). We refer to a party P_i's local variables from Figure 2 directly when clear from context.

We first assume synchrony with at most t_s corruptions. Note, as argued in Lemma 30, that all honest parties in Π_{DKG} eventually output the same value z from Π_{IT}, and thus no honest party invokes Π_{ADKG}. Then, since S is specified to perfectly simulate steps 1 to 5 of Π_{DKG} (step 1 of Figure 13), $\mathcal{A}$'s view is identically distributed to that of a run of Π_{DKG} until the condition at step 2 of Figure 13 is satisfied.

The simulation strategy after this point is essentially that of the simulator in Figure 10 of [SBKN21]. The goal is to 'hit' (in the language of [GJKR99]) the input public key y by modifying the contribution of one or more simulated parties to the final secret from the view of $\mathcal{A}$. Recall that at step 6 of Π_{DKG}, a party P_k's share of the final secret is computed as $\sum_{j \in Q} s_{jk}$ where each value s_{jk} corresponds to (the y-coordinate of) a point on a polynomial $f_j(\cdot)$ and is derived by decryption using dk_k. In particular, $\mathcal{A}$ knows, for each $j \in Q$, at most t_s points on $f_j(\cdot)$, since by the security of pke, except with nelglgible probability, the other encryptions give $\mathcal{A}$ any information about their contents. Moreover, the simulator cannot 'lie' about these points since $\mathcal{A}$ decrypted cipertexts to learn this information. Thus, S's strategy is to modify the values $(\mathsf{recon}, \dots)$ multicast by honest parties (the simulator) at line 6 of Π_{DKG} to force the public key to be exactly $y = g^x$.

Consider the first party P_i to reach step 6 of Π_{DKG} with $\mathsf{ready} = \mathsf{true}$ (step 2 of Figure 13). By t_s-intrusion tolerance of Π_{IT}, there exists an honest party P_j that correctly executes step 4 of Π_{DKG}. Note that the simulator has the state of the $\geq n - t_s$ honest parties and thus can reconstruct all polynomials $f_k(\cdot)$ where $k \in Q$ (in particular using information that P_j sent in part messages as written in Figure 13). To 'hit' $y = g^x$, the simulator needs to send recon messages consistent with a polynomial $F(\cdot)$ such that $F(0) = x$. To this end, S reconstructs such a polynomial $F(\cdot)$ in the exponent of g by interpolating t points from the 'real' secret plus the point $(0, y)$.

Note that at the end of step 6, each $P_j \in G$ sends a message of the form $M_j = (\mathsf{recon}, D_j, \pi'_j, L_j = (c^*_j, C^*_j, Q), w_j)$, where L_j and w_j are contained in part messages sent at step 4 and (due to the security of accumulator acc and Π_{IT}) cannot be changed in the simulation. For each $P_j \in G$, we therefore set $D_j = g^{F(j)}$ and use the zero-knowledge property of nizk_2 to simulate a proof that is consistent with L_j. More precisely, S simulates a proof for nizk_2 that proves knowledge of $(F(j), x^*)$ such that $D_j = g^{F(j)}$ and $C^*_j = g^{F(j)} h^{x^*}$ for some x^*. By the same argument in [SBKN21], recon messages are correctly distributed and moreover verification passes at step 7 for $\mathcal{A}$'s corrupted parties, from which secrecy follows.

We now consider t_s-uniformity, where the aim is to show, for an a priori fixed $y' \in \mathbb{G}$, the probability that the protocol run outputs y' is negligibly close to $1/p$. By t_s-correctness and t_s-consistency, all parties output a share of the secret $x = \sum_{j \in Q} x_q$, where $x_j = f_j(0)$ sampled uniformly in step 1. Note that the final secret is build from the sharings of $|Q| \geq n - t_s$ parties. Since $n - 2t_s \geq 1$, at least one honest party's contribution is included in x, and as argued the adversary has no informatione except with negligible probability about the contributions of honest parties until after Q has been fixed. Uniformity then follows from the fact that $f_j(0)$ is uniformly sampled.

Suppose now that the network is asynchronous and there are at most t_a corruptions. Suppose that one honest party outputs $z_{it} \notin \{\bot_{dkg}, \bot_{it}\}$. Then, since Π_{IT} is t_a-secure with intrusion tolerance, all honest parties will eventually output z_{it}, where z_{it} was proposed by an honest party. Since we did not use the synchrony of the network in the proof follows from above in this case. Otherwise, by t_a-security, all honest parties will invoke Π_{ADKG}. The result then follows from the (t_a, d)-security of Π_{ADKG}. □

Proof of Lemma 1. Our proof (sketch) is very similar to that of [BKL19] for Byzantine agreement, except that we rely on t_s-uniformity instead of t_s-validity to reach a contradiction. Let $t_a + 2 \cdot t_s = n$. Partition the n parties into sets S_0, S_1, S_a with $|S_0| = |S_1| = t_s$ and $|S_a| = t_a$. Consider an experiment E where communication between S_0 and S_1 is blocked by the adversary but all messages are otherwise delivered in Δ time, and two virtual copies of S_a, namely S_a^0 and S_a^1, exist that interact only with parties in $S_0 \cup S_a^0$ and $S_1 \cup S_a^1$ respectively. We construct two executions:

1. In execution 1, the network is synchronous, and parties in S_1 are corrupted and abort immediately.
2. In execution 2, the network is asynchronous, parties in S_a are corrupted and execute two independent runs of the protocol as S_a^0 and S_a^1, and all communication between S_0 and S_1 is delayed indefinitely.

In execution 1, the view of honest parties $S_0 \cup S_a^0$ is distributed identically to the views of $S_0 \cup S_a^0$ in E. Then t_s-uniformity guarantees that all parties in S_0 output a uniformly random secret. Similarly, all parties in S_1 output a secret which is uniform, and since S_0 and S_1 are disjoint, they must be independent. In execution 2, the view of honest parties $S_0 \cup S_1$ is distributed identically to $S_0 \cup S_1$ in E. But this violates t_a-consistency because, except with negligible probability, the keys output by S_0 and S_1 are different (since they are independent and uniform). □

Proof of Lemma 3. Assume the network is synchronous and there are up to t_s corruptions. The sub-protocols $\Pi_{\mathsf{BC}}^{t_s,1/2}$ and $\Pi_{\mathsf{BA}}^{t_a,t_s}$ are secure, and therefore all parties agree on the same set S. Moreover, the set has size at least $n - t_s$, since each tuple from an honest party P_i is correctly distributed via $\Pi_{\mathsf{BC}}^{t_s,1/2}$ due to validity, and by validity of $\Pi_{\mathsf{BA}}^{t_a,t_s}$, all honest parties include P_i in the set S. This implies that each A^i is an encryption of a sum of values that includes at least an honest party's value (since $n - t_s > t_s$). Moreover, the ciphertexts from the adversary are chosen independently of the ciphertext from honest parties due to the zero-knowledge proofs of plaintext knowledge, and therefore all parties compute the same tuple of encryptions of random values $A^1, \ldots, A^\ell$, with plaintexts unknown to the adversary.

Each tuple (A^i, B^i, C^i) then encodes a correct multiplication triple. This is because all parties agree on the same set S', and each B^i is an encryption of a sum of values that include an honest value, and C^i contains the product of the plaintexts from A^i and B^i due to the proof of correct multiplication.

Now assume the network is asynchronous and there are up to t_a corruptions. The only difference is that the sub-protocol $\Pi_{\mathsf{BC}}^{t_s,1/2}$ only provides weak validity. This means that the sets S or S' are not guaranteed to have size $n-t_s$, given that after time T_{bc} many of the honest parties may have obtained $\bot$. Moreover, even if any of the sets do contain $n - t_s$ parties, it is not guaranteed that all honest parties received their broadcasted values. However, any party that receives all the broadcasted values will output ℓ encrypted random multiplication triples with the plaintexts unknown to the adversary. In any other case, the output is $\bot$. □

Proof of Lemma 4. We prove each of the cases individually. We simulate in the hybrid where there is a setup generating the keys for the PKI and the threshold encryption scheme (see Definition 5). This setup can be generated using our DKG scheme from Section 3.

Case 1: Synchronous network. We describe the simulator Sim for the case where the network is synchronous and there are up to t_s corruptions.

– *Triple Generation*: Emulate the triples protocol. For that, the simulator emulates an execution of the protocol generating all the intermediate values on behalf of the honest parties. That is, for each honest party P_j, it generates random values a_j^i, $i \in [1, \ell]$ and encrypts these values and emulates all broadcasted messages (for the zero-knowledge proofs, it emulates accepted proofs). Then, it emulates the n instances of BA to agree on a set S. If the size of the set is less than $n - t_s$, then it emulates the party outputting $\perp$ in the triple generation protocol. Otherwise, it computes the values A^i for each $i \in [1, \ell]$. Similarly, the execution is emulated to possibly obtain the values B^i and C^i. If no triples have been computed for P_j, set the local variable $\mathsf{abort}_j = 1$.
– *Input Distribution*: Emulate the messages of the broadcast protocol. This means that, on behalf of each honest party, emulate the broadcast protocol using an encryption of 0 as the input. Also, emulate the $\mathcal{F}_{\mathsf{mzk}}$ functionality by outputting 1 on behalf of each honest parties, and from each corrupted party, on input $(c, (x, r))$ check that $c = \mathsf{Enc}_{\mathsf{ek}}(x, r)$ and output 1 to the adversary and 0 otherwise. The simulator waits for $\max\{T_{bc}, T_{zk}\}$. For each honest party P_j, it keeps track of the correct encrypted inputs I_j that P_j received. If the number of correct ciphertexts is less than $n - t_s$, the simulator does not compute on its ciphertexts on his behalf and sets a local variable $\mathsf{abort}_j = 1$.
– *Addition Gates*: Sim simply adds the corresponding ciphertexts locally.
– *Multiplication Gates*: Sim locally computes the ciphertexts $X \boxminus A$ and $Y \boxminus B$, and The simulator emulates the threshold decryption sub-protocol for each of these values: it sends threshold decryption shares of both ciphertexts to all parties, and outputs 1 when emulating $\mathcal{F}_{\mathsf{mzk}}$ on behalf of them. After waiting for T_{dec}, it locally computes the output ciphertext of the multiplication gate as in the protocol.
– *Output Determination*: For each party P_j, emulate the messages in the asynchronous common subset protocol with the corresponding input (either a ciphertext, which is the result of the computation, or $\perp$ in the case $\mathsf{abort}_j = 1$). If the output is a single ciphertext c, emulate the threshold decryption sub-protocol.
– *Threshold Decryption*: In a multiplication gate, simply compute the decryption shares and emulate the sending messages. In the Output Determination stage, Sim obtains the output y of the computation, and adjusts the shares such that the shares decrypt to y. In both cases, the simulator always outputs 1 on behalf of the honest parties indicating that the proofs of correct decryptions are correct.

Case 2: Asynchronous network. The only difference with respect to the case where the network is synchronous, is that the protocol BC^{t_s,t_a} only provides weak-validity. In the simulation, it implies that the simulator will also need to simulate the $\perp$ messages from the broadcast protocols, and not simulate on behalf of the honest parties which stop participating in the protocol after they aborted.

We define a series of hybrids to argue that no environment can distinguish between the real world and the ideal world.

Hybrids and security proof.

Hybrid 1. This corresponds to the real world execution. Here, the simulator knows the inputs and keys of all honest parties.

Hybrid 2. We modify the real-world execution in the zero-knowledge proofs. In the case of a synchronous network, when a corrupted party requests a proof of any kind from an honest party, the simulator simply gives a valid response without checking the witness from the honest party. In the case of an asynchronous network, the simulator is allowed to set outputs to $\perp$ as the real-world adversary.

Hybrid 3. This is similar to Hybrid 2, but the computation of the decryption shares is different. Here, the simulator obtains the output y from the ideal functionality, and if it is not $\perp$, it computes the decryption shares of corrupted parties, and then adjusts the decryption shares of honest parties such that the decryption shares $(d_1, \ldots, d_n)$ form a secret sharing of the output value y.

Hybrid 4. We modify the previous hybrid in the Input Stage. Here, the honest parties, instead of sending an encryption of the actual input, they send an encryption of 0.

Hybrid 5. This corresponds to the ideal world execution.

In order to prove that no environment can distinguish between the real world and the ideal world, we prove that no environment can distinguish between any two consecutive hybrids.

Claim 1. No efficient environment can distinguish between Hybrid 1 and Hybrid 2.

Proof. This follows trivially, since the honest parties always send a valid witness to $\mathcal{F}_{\mathsf{mzk}}$ in the case of a synchronous network. In the case of an asynchronous network, the simulator chooses the set of parties that get $\perp$ as the real-world adversary.

Claim 2. No efficient environment can distinguish between Hybrid 2 and Hybrid 3.

Proof. This follows from properties of a secret sharing scheme and the security of the threshold encryption scheme. Given that the threshold is $t_s + 1$, any number corrupted decryption shares below $t_s + 1$ does not reveal anything about the output y. Moreover, one can find shares for honest parties such that $(d_1, \ldots, d_n)$ is a sharing of y.

Claim 4. No efficient environment can distinguish between Hybrid 3 and Hybrid 4.

Proof. This follows from the semantic security of the used threshold encryption scheme.

Claim 5. No efficient environment can distinguish between Hybrid 4 and Hybrid 5.

Proof. The simulator in the ideal world and the simulator in Hybrid 4 emulate the joint behavior of the ideal functionalities exactly in the same way.

We conclude that the real world and the ideal world are indistinguishable. Finally, let us argue why the protocol has weak termination. Observe that when the protocol outputs $\perp$, parties do not terminate. This is because the protocol $\Pi_{\mathsf{acs}}^{t_s,t_a}$ does not guarantee termination, i.e. might need to run forever (see [BKL21]). However, when parties have agreement on a ciphertext to decrypt (in particular, this is the case when the network is synchronous), the threshold decryption sub-protocol ensures that honest parties can jointly collect $t_s + 1 \leq n - t_s \leq n - t_a$ decryption shares, decrypt the ciphertext and terminate. $\square$

$\mathit{\Pi}_{\mathsf{smpc}}^{t_s,t_a}(P_i)$

Let c_M be the number of multiplication gates in the circuit.

Setup. The protocol assumes a threshold additive homomorphic encryption setup, which can be generated securely using the protocol $\mathit{\Pi}_{\mathsf{DKG}}^{t_a,t_s}$ (see Section 3).

Offline Phase.

- Parties jointly run the protocol $\mathit{\Pi}_{\mathsf{triples}}^{t_a,t_s}(\ell)$ to generate $\ell = c_M$ Beaver multiplication triples.

Let x_i denote the input value of party P_i. Let $\mathsf{abort} = 0$ if a sequence of triples is received from the offline phase. Otherwise, set $\mathsf{abort} = 1$.

Input Distribution.

- P_i computes $\mathsf{TEnc}(x_i)$ and broadcasts using $\mathit{\Pi}_{\mathsf{BC}}^{t_s,1/2}$ the ciphertext $\mathsf{TEnc}(x_i)$ and uses the multi-party zero-knowledge functionality $\mathcal{F}_{\mathsf{mzk}}$ to prove knowledge of the plaintext of $\mathsf{TEnc}(x_i)$ towards all parties. Wait until $\max\{T_{bc}, T_{zk}\}$ clock ticks passed.
- If there is a broadcast or zero-knowledge proof that has not terminated, or the number of correct encryptions received is less than $n - t_s$ inputs, set $\mathsf{abort} = 1$. Continue participating in the sub-protocols, but do not compute any ciphertext.

Addition Gates. Input: $X = \mathsf{TEnc}(x), Y = \mathsf{TEnc}(y)$. Output: $Z = \mathsf{TEnc}(z)$.

- P_i locally computes $Z = X \boxplus Y$.

Fig. 14: Synchronous MPC with unanimous output (part 1).

Multiplication Gates. Input: $X = \mathsf{TEnc}(x), Y = \mathsf{TEnc}(y)$, and a triple (A, B, C). Output: $Z = \mathsf{TEnc}(z)$.

- P_i locally computes $X \boxminus A$ and $Y \boxminus B$, and sends threshold decryption shares of both ciphertexts to all parties. In addition, use the zero-knowledge functionality $\mathcal{F}_{\mathsf{mzk}}$ to prove correct decryption of the decryption shares with respect to the ciphertexts.
- Upon receiving $t_s + 1$ decryption shares with correct proofs of decryption for each of the two ciphertext, reconstruct the plaintexts $x - a$ and $y - b$.
- Compute $E = \mathsf{TEnc}((x - a) \cdot (y - b); e)$, where e is the neutral element of the randomness space. Then compute $Z = E \boxplus [(x - a)B] \boxplus [(y - b)A] \boxplus C$.
- Output Z.

Output Determination. Input x, where $x = c_i$ is the output ciphertext of the circuit if $\mathsf{abort} = 0$, and otherwise $x = \bot$.

- P_i executes the protocol $\Pi_{\mathsf{acs}}^{t_s,t_a}$ with x as input. Let S_i be the output of the protocol.
- If $S_i = \{c\}$, execute the Threshold Decryption sub-protocol on c, and after an output is given, terminate. Else, output $\bot$.

Threshold Decryption. Input: ciphertext c.

- P_i computes its decryption share s_i sends it to every other party.
- P_i proves that the value s_i is a correct decryption share of c bilaterally.
- Once $t_s + 1$ correct decryption shares are collected, send the list to every party and output the corresponding plaintext.

Fig. 15: Synchronous MPC with unanimous output (part 2).