Scalable Accountable Byzantine Agreement and Beyond

Pierre Civit
and Rachid Guerraoui
and Jovan Komatovic
and Manuel Vidigueira
EPFL
Lausanne, Switzerland
Email: firstname.name@epfl.ch

Daniel Collins
Texas A&M University
College Station, USA
Email: danielcollins@tamu.edu

Vincent Gramoli
and Pouriya Zarbafian
University of Sydney
Sydney, Australia
Email: firstname.name@sydney.edu.au

***Abstract*—No t-resilient Byzantine Agreement (or Reliable Broadcast) protocol can guarantee agreement among n correct processes in a non-synchronous network if the actual number of faulty processes f is $\geq n - 2t$. This limitation highlights the need to augment such fragile protocols with mechanisms that detect safety violations, such as forensic support and accountability.**

This paper introduces simple and efficient techniques to address this challenge by proposing a new generic transformation, $\mathcal{ABC}^{++}$. The transformation leverages two key primitives: the *ratifier* and the *propagator*. By sequentially composing these primitives with any closed-box Byzantine Agreement (or Reliable Broadcast) protocol, $\mathcal{ABC}^{++}$ produces a robust counterpart that provides both (adaptively secure) forensic support and (1-delayed adaptively-secure) accountability. The transformation incurs a subquadratic additive communication overhead, with only 1 round of overhead for decision and forensic support, and 2 additional rounds for detection in case of a safety violation (or $O\big(\log(n)\big)$ additional rounds with optimized communication).

The generality of $\mathcal{ABC}^{++}$ offers a compelling general alternative to the subquadratic forensic support solution by Sheng et al. (FC'23) tailored to HotStuff-like protocols, while being more efficient than the (strongly-adaptively-secure) quadratic $\mathcal{ABC}$ accountable transformation (IPDPS'22, JPDC'23). Moreover, it provides the first subquadratic accountable Byzantine Agreement (or Reliable Broadcast) protocols against a (1-delayed) adaptive adversary.

Finally, any subquadratic accountable Reliable Broadcast protocol can be integrated into the τ_{scr} transformation (ICDCS'22) to produce an improved variant, τ_{scr}^{++}. This new version compiles any deterministic (and even beyond) protocol into its accountable counterpart with subquadratic multiplicative communication overhead, significantly improving upon the original quadratic overhead in τ_{scr}.

1. Introduction

Byzantine agreement (BA) protocols enable a group of n processes to reach consensus even when f processes are faulty and can deviate arbitrarily from the protocol. These protocols are crucial for various distributed systems, including state machine replication (SMR), blockchain systems, and secure multi-party computation (MPC). Traditional BA protocols guarantee consensus as long as the number of faulty processes is below a certain threshold $t < n/3$. However, they fall short in scenarios where this threshold is exceeded. Typically, due to classic partitioning arguments [1], no t-resilient BA protocol can prevent disagreement during a period of asynchrony if $f \geq n-2t$ [2], [3]. This fundamental limit has motivated the community to introduce mechanisms that, at least, allow detecting faulty processes which are responsible for disagreements.

Forensic support [4], [5], [6], [7] ensures that in case of a disagreement, some correct processes will collectively hold different pieces of information such that when these pieces are combined, they provide irrefutable proof of misbehavior by a significant number of faulty processes. The collective nature of forensic support requires communication overhead to exchange and combine these pieces of evidence to simply to realize that safety has been violated, let alone to expose malicious parties. Accountability [8], on the other hand, is a stronger property. It provides irrefutable proof of misbehavior directly to every correct process, without the need to exchange and combine separate pieces of information[1]. Recent research has introduced various mechanisms to achieve these properties, but existing solutions come with notable drawbacks.

Besides not ensuring accountability, the forensic support solution of [5] is tailored specifically to a player-replaceable[2] variant of the (partially synchronous) HotStuff protocol [13], [14], and thus lacks generality and flexibility for broader use cases. On the other hand, the $\mathcal{ABC}$ transformation proposed in [2], [11] offers a more generic approach to ensure accountability, even in the presence of a strongly-adaptive adversary, but suffers from a significant

1. The original definition of accountability in [8] involves a judge-based model. Following [9], [10], [11], [12], we adopt the convention that *all* correct judges (i.e., processes) must reach a verdict.

2. We elaborate on this notion in the first paragraph of the related work section.

drawback: it incurs a quadratic communication overhead. In this scheme, each process must in particular verify a linear number of signatures, impeding scalability.

In this paper, we dissect the $\mathcal{ABC}$ transformation and introduce two new primitives: the *ratifier* and the *propagator*. These primitives enhance BA protocols by providing forensic support and accountability, with a subquadratic additive communication complexity overhead and only 1 round overhead for the decision. The ratifier allows correct processes to generate forensic pieces of evidence, tolerating a fully adaptive adversary.

The propagator ensures the dissemination of certificates to all correct nodes with accountability, while maintaining subquadratic communication by leveraging the 1-delayed-adaptive adversary model [15], where corruption decisions are postponed just long enough to allow messages in transit to be delivered.

Contributions. Let κ and λ be computational and statistical security parameters, respectively. Our contributions are as follows:

- We introduce the ratifier, a minimal functionality for forensic support. Assuming a verifiable random function (VRF) setup [16], we provide a one-round implementation that withstands fully adaptive adversaries, incurring an additional $O(\lambda n)$ message complexity overhead, where each message carries $O(\kappa)$ bits.
- We introduce the propagator, a minimal functionality for accountability, and give two alternative subquadratic implementations that tolerate a 1-delayed-adaptive adversary with either (1) $O(\lambda)$ per-process message-complexity and $O(\log(n/\lambda))$ expected rounds between a safety violation and accountable detection, or (2) $O(\sqrt{n\lambda})$ per-process message-complexity and 2 rounds between a safety violation and detection. Each message has size $O(\lambda \log n + \kappa)$ when using Jackpot [17], a concretely efficient non-interactive aggregate lottery functionality.
- We show that sequentially composing the ratifier and the propagator yields an accountable confirmer, as formalized in [11]. This primitive can then be composed with any protocol solving an easily accountable agreement task, such as Byzantine Agreement, Reliable Broadcast, or Consistent Broadcast, using the $\mathcal{ABC}$ transformation. The resulting protocol inherits forensic support, accountability, and the correctness properties of the underlying protocol Π, with complexity and security guarantees derived from the ratifier, the propagator, and Π itself. This yields the first subquadratic accountable protocols for Byzantine Agreement (and Reliable Broadcast, and Consistent Broadcast) as summarized in Table 1.
- We present the $\mathcal{ABC}^{++}$ transformation within the Accountable Universally Composable (AUC) framework [18], providing strong compositional guarantees for the broad class of easily accountable agreement functionalities, namely the UC abstractions of the easily accountable agreement tasks introduced in [11]. This class not only includes standard primitives such as Byzantine Agreement, Reliable Broadcast, and Consistent Broadcast, but also encompasses other specialized abstractions like one-to-many zero-knowledge proof, which serves as the zero-knowledge counterpart of Reliable Broadcast and is occasionally employed in MPC protocols [19], [20], [21].
- We evaluate the exact constants involved in our complexity analysis to demonstrate the practical feasibility of our solution. Additionally, we implement in Rust the cryptographic certificate logic, showing that aggregation and verification, performed once per decision, can be executed in under 50 ms, with failure probability below 2^{-40}.

TABLE 1: Comparison of Accountability and Forensic Detection Schemes. **Gen./Acc.**: Generality of supported protocols and whether accountability is ensured (Y/N). 'SMR' corresponds to a family of HotStuff-like protocols. **Comm.**: Asymptotic communication overhead. **Lat.**: Additive latency overhead. t_{acc}: Maximum number of faults for which detection is guaranteed. **Detect.**: Number of faulty processes that can be held accountable when $t < f \leq t_{acc}$. **Adapt.**: adversary adaptivity — S/W: strong / weak adaptive (with / without after-the-fact removal capabilities), D: 1-delayed adaptive.

Work	Gen./Acc.	Com.	Lat.	t_{acc}	Detect.	Adapt.
[11]	Any/Y	$+\Omega(n^2)$	$+1$	n	$n/3$	S
[22] (O)	"SMR"/Y	$+\Omega(n^3)$	$+0$	n	$n/3$	S
[22] (C)	"SMR"/Y	$\times\Omega(n)$	$+0$	n	$n/3$	S
[4]	None/N	$\Omega(n^2)$	NaN	$2n/3$	$n/3$	S
[5]	None/N	$o(n^2)$	NaN	$2n/3$	$\lambda/3$	W
$\mathcal{ABC}^{++}$	Any/Y	$+o(n^2)$	$+1$	$n-\Theta(n)$	$\lambda/3$	D

Applications. $\mathcal{ABC}^{++}$ is directly applicable to protocols whose security reduces to underlying easily accountable agreement tasks, such as those used in scalable and accountable distributed ledgers [23], state machine replication [24], [25], bulletin board services [26], [27], and payment systems [28], [29], [30]. Additionally, Accountable Byzantine Agreement can also be applied to novel problems such as rational agreement [31] and variants of the long-lasting blockchain problem [22], [32], [33], [34].

A complementary line of work focuses on general transformations that bring accountability to a wide class of distributed protocols beyond easily accountable agreement tasks. One of the earliest and most influential contributions in this direction is PeerReview [35], which provides a generic accountability layer inspired by the failure detector paradigm. In PeerReview, accountability is defined as the permanent suspicion of faulty processes, without producing externally-verifiable evidence. Later, the τ_{scr} transformation [12] strengthened this notion by providing full accountability. In this setting, violations of safety yield publicly verifiable proofs of misbehavior [8], [36], but at the cost of a quadratic multiplicative communication overhead.

We observe that the subquadratic accountable Reliable Broadcast protocols developed in this work can be directly integrated into τ_{scr} to obtain an improved transformation

τ_{scr}^{++}. This variant preserves the same accountability guarantees while reducing the communication overhead to subquadratic, as detailed in Table 2. These approaches apply to a broad class of protocols that do not naturally fall within the scope of easily accountable agreement tasks. This includes all deterministic protocols that may invoke cryptographic primitives, and even more general settings. Furthermore, previously proposed techniques to preserve hyper-properties (e.g., privacy, fairness, or unpredictability) in transformations à la PeerReview [37], [38], [39] can be naturally adapted to τ_{scr}^{++}, mitigating one of the primary limitations of such generic transformations.

TABLE 2: Comparison of General Accountability Transformations. **Com.** denotes the multiplicative communication overhead; **Ext. Ver.** indicates whether external verifiability is supported; **Adapt.** denotes the corruption model: S = strongly-adaptive, D = 1-delayed, None = static corruption.

Work	Com.	Ext. Ver.	Adapt.
PeerReview [35]	$\times o(n^2)$	N	None
PeerReview' [35]	$\times \Theta(n^2)$	N	S
τ_{scr} [11]	$\times \Theta(n^2)$	Y	S
τ_{scr}^{++} (this)	$\times o(n^2)$	Y	D

Roadmap. In §2, we review the related work. §3 defines the formal system model and the necessary preliminaries. The ratifier and its implementation are introduced in §4, followed by the propagator and its implementation in §5. Both are combined to build the generic $\mathcal{ABC}^{++}$ compiler in §6. Generalization to the Universal Composability framework is discussed in §7. §8 conducts an evaluation of our protocol's cost. Lastly, §9 concludes the work.

2. Related Work

Verifiable Random Functions (VRFs) and Player-Replaceability. VRFs [16] are cryptographic primitives that generate unpredictable yet verifiable random values, enabling efficient leader election. At each step s, processes use VRFs to determine whether they have been elected as a leader. If elected, the VRF allows the computation of a verifiable proof confirming the legitimacy of the election. In the player-replaceability paradigm [40], [41], [42], a "classic multicast" protocol, where messages are broadcasted publicly via a multicast protocol, is divided into steps that involve either all-to-all communication or leader-to-all communication. For all-to-all steps, a sublinear committee is elected using the VRF primitive to represent the group in communication. Similarly, the leader is elected using the VRF mechanism, which may sometimes elect more than one leader without harming the protocol any more than having a single malicious leader in the original protocol. To send a message in any round, a player must append a publicly verifiable proof of eligibility, and any message without a valid proof is rejected.

While this solution may seem imbalanced, the multicast primitive can be implemented using a probabilistic flooding protocol [43], which can be made secure against a 1-delayed-adaptive adversary, whose adaptive corruptions are delayed by at least one causal message delivery [15], [44], [45]. However, a broadcast protocol cannot be adaptively secure with sublinear per-process communication, as the adversary can always immediately corrupt the neighbors of the message's source.

Subquadratic Byzantine Agreement. Achieving subquadratic BA against an adaptive adversary in a non-synchronous network is highly challenging. Unlike in synchrony [46], subquadratic solutions in non-synchronous networks require a trusted setup [47], which goes beyond an ideal authentication functionality. This remains true even when private channels, Common Random Strings (CRS), and succinct non-interactive arguments of knowledge (SNARKs) are assumed to be freely available [48]. The required trusted setup seems to inherently involve correlated private randomness. Consequently, it is no big surprise that all subquadratic solutions in partially-synchrony [5], [14], [40], [48], [49], [50] or fully asynchrony [47], [51], [52] rely on the player-replaceability paradigm on top of a VRF-setup. These protocols can also be extended to efficiently handle long input values [53].

Subquadratic Byzantine Reliable Broadcast. As presented in [47], the VRF-based player-replaceability paradigm can be applied to Bracha's double-echo protocol [54], [55] to achieve subquadratic adaptively-secure performance. Multicast primitives can be implemented using the balanced flooding protocol from [15], but at the price of making the protocol secure against a 1-delayed-adaptive adversary only. Alternatively, at the cost of only providing static security, the Contagion protocol [56] achieves similar guarantees without requiring a VRF setup.

Accountable Byzantine Agreement. It has been observed that any closed-box BA object can be sequentially composed with an *accountable confirmer* to achieve accountable BA [2], [11]. The simplicity of this solution has enabled its formal verification within TLA+ [57]. This generic approach may encourage moving away from specialized solutions designed for specific BA implementations [4], [10], [58], [59], [60], even though some optimizations could avoid the extra round complexity. In this context, [22] proposed a generic transformation applicable to a broad class of HotStuff-like protocols, achieving accountability without incurring any latency overhead. However, this comes at the cost of either a $\Theta(n^3)$ additive communication overhead or a $\Theta(n)$ multiplicative overhead.

The accountable confirmer requires only one round for decision (called post-voting in OFlex [61]) and 1 additional round for the detection of a potential safety violation, and can be applied to any BA algorithm, regardless of its validity property [62]. For example, it can be used with multi-valued validated BA (MVBA) [63], [64], [65], [66], [67] or agreement on a core set (ACS) [68], [69], [70], [71], also referred to as asynchronous common subset or vector consensus. These (accountable) BA variants can then be employed to implement an (accountable) distributed ledger object (DLO), where processes agree on a (common prefix

of a) totally ordered set of transactions [23], [24], [25]. Such transactions can represent messages from an atomic-broadcast primitive or transitions in a state machine, thus solving state-machine replication (SMR). An accountable ledger can be further extended with client logic to implement an accountable *bulletin board* [26], [72], which is commonly used to describe an auditable ledger, e.g. supporting the generation of receipts for successful postings [27].

Subquadratic Accountable BA The solution of Sheng et al. [5] is the only subquadratic BA protocol that provides forensic support. However, until now, it was unclear how to extend it to an accountable protocol without incurring the linear multiplicative overhead of techniques akin to [22], thus rendering it superquadratic. Moreover, it is tailored specifically to a player-replaceable version of HotStuff, thus lacking generality and flexibility for broader use cases.

(Accountable) Finality Gadget. The notion of an (accountable) confirmer [2], [11] (see also post-voting in OFlex [61]) shares similarities with the concept of a finality gadget, a line of work that was initially defined in [73] and originated with Casper [74]. Finality gadgets, also referred to as snap-and-chat protocols [73], provide a flexible BA mechanism by combining two distinct protocols: one optimized for liveness and the other for safety. This results in two confirmation rules: a "liveness-focused" rule and a "safety-focused" rule. Subsequent improvements to the finality gadget presented in [73] have successively enhanced the functionality with predictable validity [75] and accountability [76]. Notably, it was shown in [77] that accountability implies finality.

Synchronous ID-MPC. The classical impossibility of fairness in secure multi-party computation (MPC) when fewer than $n/2$ processes are correct [78] has motivated the study of synchronous MPC with identifiable abort (ID-MPC) [79], [80], [81]. This line of work, originally explored under the notion of covert security [82], [83], culminated in the publicly accountable MPC protocol of [84]. While our setting does not aim to preserve input or output privacy—a central concern in ID-MPC—we focus on asynchronous systems, where omission-sending faults are indistinguishable from network delays, making reliable detection fundamentally more difficult. As such, the two lines of research pursue different goals under different assumptions, and are thus largely incomparable and orthogonal.

3. Preliminaries

Notations. Notations are summarized in Table 3.

Processes and Network. We consider a static set of n processes $\Psi = \{p_1, \ldots, p_n\}$. A distributed protocol is defined as the tuple $\Pi = (\Pi_1, \ldots, \Pi_n)$, where protocol Π_i is prescribed to process p_i. An adversary (elaborated upon below) can adaptively corrupt processes. Processes that remain uncorrupted are referred to as *correct*, and they execute their respective state machines as prescribed by the protocol. Communication occurs over a reliable, authenticated, point-to-point network, meaning that every pair of correct processes can communicate, and a message sent by a correct process to another correct process is eventually delivered. Additionally, the receiver can always identify the sender. We consider a fully-asynchronous network, without upper bound on message delays, but our transformation can apply to (closed-box) protocols that assume partial synchrony [85], where after some unknown Global Stabilization Time (GST), message delays are bounded by a known constant.

TABLE 3: Summary of Notations.

Notation	Description
n	Number of (#) processes.
f	Actual number of failures.
$\epsilon, \delta, \hat{\delta} > 0$	Small constants for Chernoff bounds
$t = \lceil n(\frac{1}{3} - \epsilon) \rceil - 1$	# tolerated failures (nominal mode)
$t_{acc} = n - \Theta(n)$	# tolerated failures (degraded mode)
$\gamma = \frac{n - t_{acc}}{n}$	Fraction of ensured correct processes.
λ	Expected size of a committee.
κ	The computational security parameter.
$W_{\delta}^{\epsilon,\lambda} = (\frac{2}{3} + \epsilon)(1 - \delta)\lambda$	Size of a VRF-quorum.
$B_{\delta,\hat{\delta}}^{\epsilon,\lambda} = 2W_{\delta}^{\epsilon,\lambda} - (1 + \hat{\delta})\lambda$	Size of VRF-quorums' intersection.

Adversary. The adversary is modeled as a state machine that interacts with both the network and the processes. The adversary is assumed to be computationally bounded and unable to break the cryptographic primitives introduced later in the protocol. The adversary controls message scheduling, deciding when messages in transit are delivered. The adversary can corrupt processes dynamically, with f denoting the total number of corruptions. If no restriction is specified, the adversary is adaptive. We also consider 1-delayed-adaptive adversaries [15] (delayed-adaptive for short), where a corruption decision is delayed by the (unkown) maximum time required to deliver a message. Specifically, when a so-far correct process p_i sends a message m to a so-far correct process p_j, the adversary cannot corrupt p_j before it receives and potentially forwards m. Corrupted processes, called Byzantine, are coordinated arbitrarily by the adversary. An adversary defines a probabilistic space over the executions of the global system. The probability is taken over the random coins of the parties, the random coins of the adversary, and the sampling coins from the setup distribution.

We introduce two thresholds: an *optimistic* threshold $t \in [0, n(\frac{1}{3} - \epsilon))$ for some arbitrarily small constant $\epsilon \in \Omega(1)$, and a *pessimistic* threshold $t_{acc} \in [0, 1 - \Theta(n)]$. We note $\gamma = (n - t_{acc})/n$, the ratio of processes that is assumed to be correct even in bad scenarios. The protocol will be required to solve BA when $f \leq t$, provide forensic support for any f, and ensure accountability when $f \leq t_{acc}$.

Byzantine Agreement. BA protocols allow n processes to agree on a common value even in the presence of faulty processes that may behave arbitrarily. The BA's specification is given in the Module 1 presented below.

Module 1 Byzantine Agreement $\langle val, \rho \rangle$

Parameters:
- A validity property val. ▷ Details can be found in [62]
- $\rho : f \in [1 : n] \mapsto [0, 1]$, the minimum probability of success

Events:
- *request* $\mathsf{propose}(v_i \in \mathsf{Value})$: a process inputs a value v_i
- *notification* $\mathsf{decide}(v_o \in \mathsf{Value})$: a process outputs a value v_o.

Properties:
Let f be the actual number of failures. With probability at least $\rho(f)$, the following is ensured:
- *Termination:* All honest processes eventually decide on a value.
- *Agreement:* No two honest processes decide on different values.
- *Validity:* Any decided value is valid according to val.

The module is parametrized with a validity property val and function ρ. A validity property, formalized in [62], maps any input configuration, representing the proposals of the correct processes, with a set of admissible values. For example, the strong unanimity validity property states that if all correct processes unanimously propose a common value, then only that value can be decided. The function ρ maps the actual number of failures f with the corresponding guaranteed probability of success. Let us note that ρ is non-increasing and $\rho(\lceil (n/3) \rceil) = 0$. Classically, ρ is associated with a threshold $t' < n/3$, such that $\rho(t') = 1 - o(1)$, while nothing is guaranteed for $\rho(t' + 1)$.

Complexity. We focus on the communication and round complexities of BA under asynchronous and partially synchronous conditions. For each execution, the bit complexity refers to the number of bits sent by correct processes, while the round complexity measures how long the decision process takes based on message delays. In the partially synchronous model, the time and bits exchanged before Global Stabilization Time (GST) are not counted. The expected bit and round complexities of any BA protocol are defined as the worst-case expectations over all adversarial strategies.

Forensic support and accountability. We adopt the nomenclature of accountability outlined in [8], [18]. In this framework, the breach of a specific targeted safety property should result in a *verdict*, identifying the parties responsible for the security violation. Verdicts are computed by *judges* based on *pieces of evidence*. We focus exclusively on *judges* that are *fair* and *public*. A public judge relies solely on publicly verifiable evidence, that can be published by individual parties, allowing external observers to access and validate them, thereby enabling *external verifiability* [8], [18]. A fair judge produces, with all but negligible probability, *fair* verdicts—verdicts that do not blame honest parties adhering to the protocol. Furthermore, we consider *individual accountability*, where verdicts take the form $\mathsf{dis}(D)$, meaning that all processes in the set D are deemed dishonest. Each correct process p_i runs a local instance J_i of such a (fair and public) judge.

Jumping slightly ahead, the judge we consider operates on two triplets, π_1 and π_2, where each $\pi_i = (m_i, Q_i, \sigma_i)$ for $i \in \{1, 2\}$. Here, $Q_i \subseteq \Psi$ represents a subset of processes, m_i is a message, and σ_i is a valid multi-signature certifying that *all* members of Q_i have signed m_i (see next section). If these signatures are valid and m_1 and m_2 are *conflicting*, meaning they have the same header but different payloads (referred to as *mutant messages* in [86]), then the judge outputs the verdict $\mathsf{dis}(Q_1 \cap Q_2)$. This verdict is fair, since a correct process will never sign conflicting messages.

Forensic support ensures that if a specific safety property is violated, correct processes possess evidence that, when presented to a fair judge, leads to the identification of malicious parties. Accountability strengthens this by eliminating the need for centralized evidence collection. Each correct process independently holds self-verifiable proof, allowing it to act as the judge or directly forward the proof to any external party (client), who can also serve as the judge.

Concretely, let Π be a distributed protocol equipped with some output $\mathsf{yield\_certificate}(s \in \mathsf{String})$ or $\mathsf{generate\_proof}(s \in \mathsf{String})$, and J a fair judge (for Π). Adapting the presentation of [5], we say that Π provides $(m, k, d, J, \mathsf{valid}(\cdot))$-*forensic support for safety property* P, if the violation of P with $f \leq m$ implies, except with negligible probability, the existence of a set of k honest processes $(p_{j_1}, ..., p_{j_k})$ that have triggered $\mathsf{yield\_certificate}(s_1)$, ..., $\mathsf{yield\_certificate}(s_k)$ respectively, such that $J(s_1 :: \ldots :: s_k)$ returns a set of (undeniably malicious) d distinct processes, while $\mathsf{valid}(s_j) = true$ for each $j \in [1..k]$. The role of the valid predicate can be ignored for now, but will take on its full meaning when we look at the composition of the ratifier and the propagator. Let us note that this definition does not say how the pieces of evidence s_1, ..., s_k are collected by the judge. We say that Π provides (m, d, J)-*accountability for safety property* P, if the violation of P with $f \leq m$ implies that, except with negligible probability, every correct process $p_i \in \Psi$ eventually triggers $\mathsf{generate\_proof}(s_i)$ such that $J(s_i)$ returns a set of (undeniably malicious) d distinct processes. We call *detection-round complexity*, the expected number of rounds that separate a safety violation from its detection.

3.1. Cryptographic Primitives

We now present the cryptographic primitives assumed by our protocol, including hash functions, multi-signature schemes, verifiable committee sampling, and public key infrastructure. Let κ be the security parameter.

Hash. We assume a collision-resistant hash function hash that maps any value into a sequence of $O(\kappa)$ bits.

Multi-signature. We assume the existence of a dynamic accountable non-interactive multi-signature scheme [87], [88], [89], [90], [91] (DANMSS) defined as a tuple of polynomial-time algorithms

$$\mathsf{DANMSS} = (\mathsf{setup}, \mathsf{kg}, \mathsf{sign}, \mathsf{cb}, \mathsf{ver})$$

with the following functionalities:
- $\mathsf{setup}(1^\kappa) \rightarrow \mathsf{pp}$: On input a security parameter κ, outputs public parameters pp. These parameters are implicitly provided to all other algorithms.

- $\mathsf{kg}() \rightarrow (\mathsf{pk}_i, \mathsf{sk}_i)$: Key generation outputs a fresh public/secret key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$ for signer p_i.
- $\mathsf{sign}(\mathsf{sk}_i, m) \rightarrow \sigma_i$: Signs a message $m \in \{0,1\}^{poly(\kappa)}$ using secret key sk_i, yielding a signature σ_i.
- $\mathsf{cb}(m, \{(\mathsf{pk}_i, \sigma_i)\}_{p_i \in G}) \rightarrow \sigma / \perp$: Combines a set of individual signatures from a group of signers G on the message m into a single signature σ, or returns $\perp$ if verification fails.
- $\mathsf{ver}(m, \sigma, \{\mathsf{pk}_i\}_{i \in G}) \rightarrow \{0,1\}$: Verifies the (potentially combined) signature σ on message m with respect to the group G of public keys.

The scheme satisfies the following security guarantees, except with negligible probability:

- *Unforgeability* No (PPT) adversary can forge a valid aggregate signature on a message m under a set of public keys, unless it queried the signing oracle for each key.
- *Robustness* The scheme guarantees correct behavior in the presence of potentially adversarial signers, as follows:
  1) The verification algorithm $\mathsf{ver}(m, \sigma_i, \mathsf{pk}_i)$ accepts any signature σ_i honestly produced via $\mathsf{sign}(\mathsf{sk}_i, m)$ for any $i \in [n]$.
  2) For any message m and set G of public keys, if each $(\mathsf{pk}_i, \sigma_i)$ passes individual verification, then the combination $\mathsf{cb}(m, \{(\mathsf{pk}_i, \sigma_i)\}_{i \in G})$ outputs a multi-signature σ that is accepted by $\mathsf{ver}(m, \sigma, \{\mathsf{pk}_i\}_{i \in G})$.

  This property ensures that even in the presence of malicious signers, an honest aggregator can compute a valid aggregate signature as long as the individual signatures verify correctly.

Public Key Infrastructure (PKI). We assume a (synchronous) bulletin-board PKI model, where each party independently generates its private signing key(s) and posts the corresponding public verification key(s) to a public bulletin board. The adversary may adaptively corrupt parties and, upon corruption, may modify their keys based on the complete public setup—including the verification keys of honest parties and any available common reference string (CRS), if present. The synchronous aspect of the bulletin-board PKI lies in the existence of a fixed time τ_0, after which any missing publication is definitively interpreted as the absence of a key (i.e., mapped to $\perp$). Hence, any correct process must publish its public key(s) before τ_0 to avoid being excluded. The (synchronous) bulletin-board PKI model is equivalent to allowing parties to register arbitrary strings as public keys with the ideal certification authority $\mathcal{F}_{\mathsf{CA}}$ [92] before the starting time τ_0 of the online protocol.

Verifiable Committee Sampling. Using VRFs and a (synchronous) bulletin-board-PKI, it is possible [48] to implement (with a confined bias) validated committee sampling (VCS), which is a primitive that allows processes to elect committees without communication and later prove their election. We follow the presentation of Cohen et al. [93]. VCS provides every process p_i with a private function $\mathsf{sample}_i(s, \lambda)$, which gets a string s (typically the label of some specific step) and a threshold $\lambda \in [1 : n]$ and returns a tuple $\langle v_i, \sigma_i \rangle$, where $v_i \in \{true, false\}$ and σ_i is a proof (of size $O(\kappa)$) that $v_i = \mathsf{sample}_i(s, \lambda)$. If $v_i = true$ we say that p_i is sampled to the committee for s and λ. The primitive ensures that p_i is sampled with probability λ/n. In addition, there is a public (known to all) function, $\mathsf{committee\_val}(s, \lambda, i, \sigma_i)$, which gets a string s, a threshold λ, a process identification i and a proof σ_i, and returns *true* or *false*. Finally, we note $C(s, \lambda) = \{p_i \in \Psi | \mathsf{sample}_i(s, \lambda) \in \{\langle true, \sigma \rangle | \sigma \in \{0,1\}^*\}\}$. Intuitively, $C(s, \lambda)$ corresponds to the committee (secretly) elected for step s. Consider a string s. For every $p_i \in \Psi$, let $\langle v_i, \sigma_i \rangle$ be the return value of $\mathsf{sample}_i(s, \lambda)$. The following is satisfied:

- (Correctness) $\mathsf{committee\_val}(s, \lambda, i, \sigma_i) = v_i$.
- (Unpredictability) If p_i is correct, then it is infeasible to compute $\mathsf{sample}_i(s, \lambda)$.
- (Unforgeability) It is infeasible to find $\langle v, \sigma \rangle$ s.t. $v \neq v_i$ and $\mathsf{committee\_val}(s, \lambda, i, \sigma_i) = true$

We fix $\delta, \hat{\delta} \in \Omega(1)$ as arbitrarily small positive constants, which capture deviation parameters in the relevant Chernoff bounds, and define:

- $W_\delta^{\epsilon,\lambda} = (\frac{2}{3} + \epsilon)(1 - \delta)\lambda$: conservative size of a "VRF-quorum".
- $B_{\delta,\hat{\delta}}^{\epsilon,\lambda} = 2W_\delta^{\epsilon,\lambda} - (1 + \hat{\delta})\lambda$: conservative size of the intersection of two VRF-quorums.

For readability, we may write W (resp., B) as shorthand for $W_\delta^{\epsilon,\lambda}$ (resp., $B_{\delta,\hat{\delta}}^{\epsilon,\lambda}$), when the parameters δ, $\hat{\delta}$, ϵ, and λ are clear from context.

The following results, based on standard Chernoff bounds, represent VRF-based analogs of quorum system properties:

Theorem 1. Let $\mathsf{H}(s, \lambda)$ denote the random variable that returns the set of (so-far honest) processes p_i such that the sampling event $\mathsf{sample}_i(s, \lambda)$ is not preceded by the corruption of p_i in the execution, and let $C(s, \lambda) = \{p_i \in \Psi | \mathsf{sample}_i(s, \lambda) = \langle true, * \rangle\}$, sampled uniformly at random with expected size λ. [3]

- **Liveness.** With $W = (1 - \delta)(\frac{2}{3} + \epsilon)\lambda$, we have:

$$\Pr\left[|C(s, \lambda) \cap \mathsf{H}(s, \lambda)| \leq W\right] \leq \exp\left(-\frac{\delta^2(2 + 3\epsilon)}{6}\lambda\right)$$

- **Forensics.** Let $Q, Q' \subseteq C(s, \lambda)$ with $|Q| = |Q'| = W_\delta^{\epsilon,\lambda}$. Let $B = 2W_\delta^{\epsilon,\lambda} - (1 + \hat{\delta})\lambda$ Then:

$$\Pr\left[|Q \cap Q'| \leq B\right] \leq \exp\left(-\frac{\hat{\delta}^2}{2 + \hat{\delta}}\lambda\right)$$

The Liveness result shows that if a set $C(s, \lambda)$ is assigned a task, with overwhelming probability, a "VRF-quorum" of size $W \approx 2\lambda/3$ will perform it. This mirrors the quorum availability property in classic quorum systems. The Forensic result ensures that if two "VRF-quorums" perform some task, their intersection will contain $\approx \lambda/3$ (accountable) processes, providing the analog of quorum intersection in classic quorum systems.

3. Here, Pr represents the worst-case probability over adversarial strategies with up to t adaptive corruptions (no liveness guarantees when $f > t$).

4. Ratifier

After some (unreliable) BA instance yielding the (pre-)decision v, processes want to answer the question: "Do we agree that we have the same pre-decided value v?". Moreover, we want to ensure forensic support in the event of disagreement. We capture these requirements through the *ratifier* primitive, which is specified in Module 2. Naively, this can be implemented by a single round of all-to-all communication where parties sign and broadcast their value v. However, this incurs quadratic communication overhead, which is undesirable in large-scale deployments.

Instead, in our protocol, a VRF-sampled committee of expected size λ broadcasts a SUBMIT message containing (a hash of) their signed pre-decided value to all other processes. Upon receiving sufficiently many ($W \simeq 2\lambda/3$) valid SUBMIT messages supporting the (hash of) pre-decision v, a process can confirm its decision on v and store the signed messages from a "VRF-quorum" of size W for future auditing. When a DANMSS scheme [90] is used, those signatures can be combined into a multi-signature σ, to produce a "VRF-quorum certificate" $(\mathsf{hash}(v), Q, \sigma)$. This certificate proves that all the members of Q signed (the hash of) v. When this is sequentially composed with a (closed-box) t'-resilient BA protocol, all properties hold as long as $f \leq \mathsf{min}(t, t')$, as guaranteed by the first Chernoff bound in Theorem 1.

In the event of a disagreement, two processes will store conflicting certificates (h, Q, σ) and (h', Q', σ'), for conflicting (hashes of) values. These two certificates form undeniable proof of misbehavior against the processes in $Q \cap Q'$ that signed conflicting SUBMIT messages. With overwhelming probability, the number of such provably guilty processes is approximately $\lambda/3$, as supported by the second Chernoff bound in Theorem 1. This yields Theorem 2.

Module 2 Ratifier

Parameters:
- Integer $t = \lceil (n(\frac{1}{3} - \epsilon)) - 1 \rceil$
- Predicate(String $\rightarrow \{true, false\}$) valid$(\cdot)$

Events:
- *request* submit($v \in$ Value): a process submits a value v_i
- *notification* confirm($v_o \in$ Value) a process confirms a value v_o
- *notification* yield_certificate($cert \in$ String) a process outputs a certificate.
  $cert$ will be interpreted as an element of Certificate by the judge J_{rat} for the forensic support in case of disagreement

Properties:
- *Integrity:* For any $f \in [0 : n]$, a process cannot confirm a value that it did not submit.
- *Optimistic Convergence:* If $f \leq t$ and all correct process submit the same value v, every correct process eventually confirms v.
- *Validity:* A process only yields valid certificates w.r.t. valid$(\cdot)$

Forensic-Properties:
- *Agreement:* No two correct processes output different values.
  This property does not have to be ensured (if preconditions of optimistic convergence are not met), but forensic support (defined in §3) will be then required in case of disagreement.

Algorithm 2 Π_{rat} - Pseudocode (for process p_i)

1: **Parameters:**
2: String $step$
3: Real $\lambda, \epsilon, \delta$ ▷ See parameters in Table 3.

4: **Local variables:**
5: Dictionary(Processes, Signatures) $signatures_i \leftarrow \perp$
6: Dictionary(Processes, VRFproofs) $\pi_i^{poe} \leftarrow \perp$
7: Value $predecided_i \leftarrow \perp$
8: Hash_Value $h_i \leftarrow \perp$

9: **upon** submit($v \in$ Value):
10: $(predecided_i, h_i) \leftarrow (v, \mathsf{hash}(v))$
11: $\langle elected_i, \sigma_i^e \rangle \leftarrow \mathsf{sample}(step, \lambda)$
12: **if** $elected_i$:
13: $\sigma_i^s \leftarrow \mathsf{sign}_i(\langle step, \text{SUBMIT}, h_i \rangle)$
14: **broadcast** $\langle step, \text{SUBMIT}, h_i, \sigma_i^e, \sigma_i^s \rangle$

15: **upon** $m_j = \langle step, \text{SUBMIT}, h_j, \sigma_j^e, \sigma_j^s \rangle$ is received from process p_j and $h_j = h_i \neq \perp$:
16: **if** committee_val$(step, \lambda, j, \sigma_j^e) \wedge$ verify$(\langle step, \text{SUBMIT}, h_j \rangle, \sigma_j^s, pk_j)$:
17: $signatures_i[p_j] \leftarrow \sigma_j^s$
18: $\pi_i^{poe}[p_j] \leftarrow \sigma_j^s$

19: **upon** $|Q| \geq W_\delta$ with $Q = \{p_j \in \Pi, signatures_i[p_j] \neq \perp\}$:
20: $\sigma^Q \leftarrow \mathsf{cb}(\langle step, \text{SUBMIT}, h_i \rangle, \{pk_j, signatures_i[p_j]\}_{j \in Q})$
21: **trigger** yield_certificate$((step, h_i, Q, \sigma^Q), \pi_i^{poe})$
Although the proof of eligibility π_i^{poe} for VRF-quorum Q is disregarded by the judge J_{rat}, it will be utilized by the propagator subprotocol, as detailed in the next section.
22: **trigger** confirm$(predecided_i)$

Theorem 2. Let $(J_{rat}, \mathsf{valid\_full})$ be the pair judge-predicate presented in Algorithm 3. The judge J_{rat} is fair for Π_{rat} (see Algorithm 2), which implements the Ratifier module (see Module 2) with t-resiliency, an expected communication complexity of $O(\lambda n \kappa)$, and 1 round, and provides $(n, 2, B_{\delta,\hat{\delta}}^{\epsilon,\lambda}, J_{rat}, \mathsf{valid\_full})$-forensic support for the agreement property, stating that no two correct processes confirm two different values.

PROOF SKETCH. First, we prove that the protocol Π_{rat} implements the Ratifier module with t-resiliency. The integrity property is trivial. Assume $f \leq t$ and $\exists v \in$ Value, such that all correct processes trigger submit(v). By Theorem 1, the probability that fewer than $W_\delta^{\epsilon,\lambda}$ so-far correct processes set their variable $elect_i$ to $true$ line 11 is bounded by: $\Pr[|C(s,\lambda) \cap \mathsf{H}(s,\lambda)| \leq W_\delta^{\epsilon,\lambda}] \leq neg(\lambda)$. Thus, with overwhelming probability, at least $W_\delta^{\epsilon,\lambda}$ correct processes send a (justified) SUBMIT message for the (hash of) value v, implying that line 14 is reached by at least $W_\delta^{\epsilon,\lambda}$ correct processes. Consequently, with overwhelming probability, line 17 is reached by every correct process p_i at least $W_\delta^{\epsilon,\lambda}$ times, leading to the confirmation of v at line 22.

The expected communication complexity is derived as follows. Let X_j be a Bernoulli random variable indicating whether $p_j \in C(s,\lambda)$, where $\Pr[X_j = 1] = \lambda/n$. Define $Z_L = \sum_{p_j \in \mathsf{H}} X_j$. By linearity of expectation, $\mathbb{E}(Z_L) \leq \lambda$. Therefore, the expected number of correct processes broadcasting a SUBMIT message at line 14 is λ. Since each correct VRF-committee's member broadcasts its

Algorithm 3 Fair judge J_{rat} for Π_{rat} (see Algorithm 2)

Helper Definition:

We define valid_light the predicate that parses a string as a tuple $(step, h, Q, \sigma) \in \mathsf{String} \times \mathsf{Hash\_Value} \times \mathsf{Set(Process)} \times \mathsf{MultiSignature}$, and returns true if and only if (a) $|Q| \geq W$ and (b) σ is a valid multi-signature of the message $m_h = \langle step, \text{SUBMIT}, h \rangle$ from the subset Q of processes, s.t. $\mathsf{ver}(m_h, \sigma, \{pk_j\}_{p_j \in Q})$ returns true. LCertificate denotes the corresponding type of strings c such that $\mathsf{valid\_light}(c) = true$, which we call *lightweight certificates*. If $c = (step, h, Q, \sigma) \in \mathsf{LCertificate}$, we let $c.\mathsf{step} = step$, $c.\mathsf{hash} = h$, and $c.\mathsf{quorum} = Q$.

We denote by $\mathsf{conflict}_{rat}$ the predicate over pairs of certificates that returns *true* if $c_1.\mathsf{hash} \neq c_2.\mathsf{hash}$ and $c_1.\mathsf{step} \neq c_2.\mathsf{step}$, and *false* otherwise. A tuple $(c_1, c_2) \in \mathsf{LCertificate}^2$ is a *proof of misbehavior* if $\mathsf{conflict}_{rat}(c_1, c_2)$. We denote by Proof the corresponding type.

A *full certificate* is a pair (c, π^{poe}), such that (1) $c = (step, h, Q, \sigma) \in \mathsf{LCertificate}$, and (2) (c, π^{poe}) proves the eligibility of processes in Q for step $step$, i.e. we can define the predicate valid_full that maps pairs of the form (c, π^{poe}) to true if (a) $\mathsf{valid\_light}(c) = true$, and (b) $\forall p_j \in G, \mathsf{committe\_val}(step, \lambda, j, \pi^{poe}[p_j]) = true$. FCertificate denotes the corresponding type of strings c such that $\mathsf{valid\_full}(c) = true$. If $\tilde{c} = (c, \pi^{poe}) \in \mathsf{FCertificate}$, we note $\tilde{c}.\mathsf{step} = c.step$, $\tilde{c}.\mathsf{hash} = c.\mathsf{hash}$, $\tilde{c}.\mathsf{quorum} = c.\mathsf{quorum}$, $\tilde{c}.\mathsf{poe} = \pi^{poe}$, and $\tilde{c}.\mathsf{light} = c$.

upon $\mathsf{input}(string \in \mathsf{String})$:
  parse *string* as a proof of misbehavior (c_1, c_2)
  if the parsing has succeeded, i.e. if $(c_1, c_2) \in \mathsf{Proof}$:
    return $\mathsf{dis}(c_1.\mathsf{quorum} \cap c_2.\mathsf{quorum})$
  else:
    return $\mathsf{dis}(\emptyset)$

SUBMIT message of size $O(\kappa)$ to all processes, the overall communication complexity is $O(\lambda \kappa n)$.

Second, the fairness of J_{rat} follows directly from the fact that no correct process signs conflicting SUBMIT messages.

Third, we prove the forensic support property. Assume agreement is violated, i.e., there exist two correct processes p_i and p_j that have confirmed different values v_i and v_j, with $v_i \neq v_j$. The condition in line 19 implies that line 17 was reached by process p_i (respectively, p_j) at least $W_\delta^{\epsilon,\lambda}$ times due to the reception of SUBMIT messages signed by distinct sets of processes Q_i (respectively, Q_j). Therefore, processes p_i and p_j have triggered $\mathsf{yield\_certificate}((c_i, *))$ and $\mathsf{yield\_certificate}((c_j, *))$, respectively, with $c_i = (step, \mathsf{hash}(v_i), Q_i, \sigma^{Q_i})$ and $c_j = (step, \mathsf{hash}(v_j), Q_j, \sigma^{Q_j})$. Hence, $J_{rat}(c_i, c_j)$ returns $Q_i \cap Q_j$, while, by Theorem 1 (Forensic), $\Pr[|Q_i \cap Q_j| \leq B_{\delta,\hat{\delta}}^{\epsilon,\lambda}] \leq neg(\lambda)$.

Finally, if a so-far correct process p_i triggers $\mathsf{yield\_certificate}(\tilde{c}_i)$, the checks at line 16, and 19 ensures that $\mathsf{valid\_full}(\tilde{c}_i)$, which concludes the proof. □

5. Propagator

As discussed in §1 and §3, forensic support does not specify how the judge acquires the necessary pieces of evidence to detect misbehavior. This task is handled by the *propagator*, defined in Module 3. A straightforward implementation would be for every correct process to broadcast its certificate, but this leads to quadratic communication complexity due to the all-to-all broadcast pattern.

Module 3 Propagator

Parameters:
$\mathsf{Predicate}(\mathsf{String} \to \{true, false\})$ $\mathsf{valid}(\cdot)$
$\mathsf{Predicate}(\mathsf{String}^2 \to \{true, false\})$ $\mathsf{conflict}(\cdot, \cdot)$

Events:
- *request* $\mathsf{propagate}(c)$ with $\mathsf{valid}(c) = true$: a process propagates a valid certificate c
- *request* $\mathsf{generate\_proof}(\pi \in \{0,1\}^*)$: a process outputs a proof of misbehavior.

Accountability-Properties:
- *Conflict-freeness:* No two correct process processes p_i, p_j propagate c_1 and c_2 respectively, such that $\mathsf{conflict}(c_1, c_2)$.
  Accountability (defined in §3) will be required in case of violation of this property.

An alternative might involve leveraging the (Byzantine) Probabilistic Quorum System property [94, Lemma 4.5]. This lemma states that if two correct processes p_i and p_j independently sample two probabilistic quorums Q_i and Q_j, where $|Q_i| = |Q_j| = \lambda\sqrt{n}$, and the adversary can only independently corrupt $n - \Theta(n)$ processes, then with probability $1 - neg(\lambda)$, $Q_i \cap Q_j$ contains at least one correct process p_k. This process could receive both potentially conflicting certificates and raise an alarm by broadcasting the corresponding proof to all parties. While this solution is simple and appealing, it has a critical drawback: once p_i broadcasts its certificate $cert_i$ to Q_i, the adversary could (adaptively) corrupt Q_i before p_j receives its own certificate. Therefore, this propagation method only provides static security.

Similarly, one might consider electing a VRF-based committee C responsible for monitoring and disseminating potentially conflicting certificates. Each process would forward its confirmed certificate to C, which would then raise an alert upon detecting a conflict. However, this approach suffers from the same core limitation. Once a correct process p_i sends its certificate $cert_i$ to C, the adversary could adaptively corrupt all members of C before another correct process p_j obtains and forwards a conflicting certificate $cert_j$. In such a case, no honest party ever learns both certificates, and the inconsistency goes undetected. Importantly, this vulnerability persists even under a 1-delay-adaptive adversary. The issue is not merely the speed of adaptive corruptions, but the lack of a mechanism to ensure that conflicting certificates are delivered to the monitoring committee with a fixed temporal gap. Without such synchronization between p_i and p_j, the adversary can always exploit timing to maintain plausible deniability.

That said, one might suspect that the problem inherently admits a quadratic lower bound. Indeed, in the pessimistic case, where a network partition or adversarial scheduling causes disagreement, every process must prepare for the possibility of conflicting certificates. Two correct processes

p_i and p_j, unaware of each other's identities, must somehow reach a third process p_k (possibly even p_i or p_j themselves) to serve as a relay: a party that receives both certificates and disseminates proof of disagreement. The challenge is that such a relay can be corrupted almost immediately (after just one causal hop) before it sees both certificates. This suggests that the only robust solution is for every process to broadcast its certificate to all others, ensuring that there are sufficiently many (more than t_{acc}) honest candidates available to act as relays, even under aggressive adaptive corruptions.

The key observation is that, in the nominal case, all correct processes are expected to confirm the same value. Thus, their collective flooding of certificates can be highly optimized: there is no need to forward redundant copies of the same message. While this optimization is not apparent in a classical broadcast pattern, where the sender immediately transmits to all peers, it becomes obvious in a flooding pattern, where each process forwards messages only to locally sampled neighbors. If a process has already forwarded a message for (the hash of) a value v, originating from some source p_i, it need not forward the same message again just because it later sees it endorsed by a different source p_j.

This leads us to employ $\Pi^{prop}_{\mathsf{ERFlood}}$ (Algorithm 4), a minor modification of the flooding protocol Π_{ERFlood} described in [15]. The protocol's pseudocode is simple (though its analysis is far from trivial): upon receiving a verifiably valid message m, each party flips a coin for each other process that decides, with some probability ρ^*, if m should be relayed to this process. Specifically, the protocol Π_{ERFlood} ensures that every process relays messages to a different random subset of processes, whose size follows a binomial law $\mathsf{Bin}(n-1, \rho^*)$. By making this subset large enough, Π_{ERFlood} guarantees the formation of a connected Erdős–Rényi graph with a low diameter.

In $\Pi^{prop}_{\mathsf{ERFlood}}$, which is based on Π_{ERFlood}, a message m containing a valid certificate c is relayed by process p_i only if it has not already relayed m *or* another message m' with a certificate c' where $c'.\mathsf{hash} = c.\mathsf{hash}$.

This approach, however, remains vulnerable to a subtle attack. In the absence of committee eligibility proofs, the adversary can fabricate $\Theta(n/W)$ conflicting *lightweight certificates* (defined in Algorithm 3), each signed by disjoint sets of non-elected signers of size W. These fake certificates, lacking any overlap, cannot be linked to any Byzantine process. By flooding the network with such certificates, the adversary induces a $\Theta(n/W)$ multiplicative communication overhead, effectively reverting the protocol to quadratic complexity. Conversely, if correct processes refrain from relaying all certificates to reduce load, they risk omitting a genuinely conflicting pair of honestly generated certificates, thereby undermining accountability.

Hence, $\Pi^{prop}_{\mathsf{ERFlood}}$ propagates full certificates, where the corresponding proof of eligibility avoids the aforementioned attack. More formally:

Definition 1. let valid be a predicate over strings, and conflict a predicate over pairs of strings, such that not being conflicting ($\mathsf{conflict}(c, c') = \mathit{false}$) is an equivalence relation. We say that the pair $(\mathsf{valid}, \mathsf{conflict})$ is *propagation-friendly with respect to* (J, d) if $\mathsf{valid}(c_1) = \mathsf{valid}(c_2) = \mathit{true} \wedge \mathsf{conflict}(c_1, c_2) = \mathit{true}$ implies that the judge $J(c_1, c_2)$ returns verdict $\mathsf{dis}(G)$ with $|G| \geq d$, except with negligible probability.

The propagation-friendliness condition guarantees that if two apparently valid and conflicting certificates exist, then they necessarily implicate at least d common signers. While this property would not hold for lightweight certificates, it is recovered when full certificates include proofs of eligibility.

Observe that $(\mathsf{valid\_full}, \mathsf{conflict}_{rat})$, defined in Algorithm 3, is propagation-friendly with respect to $(J_{rat}, B^{\epsilon,\lambda}_{\delta,\hat{\delta}})$. Then, it is easy to see that the analysis of $\Pi^{prop}_{\mathsf{ERFlood}}$ mirrors the (non-trivial) analysis of Π_{ERFlood} from [15].

Algorithm 4 $\Pi^{prop}_{\mathsf{ERFlood}}\langle\rho^*\rangle$ - Pseudocode (for process p_i)

Parameters
  Real ρ^* ▷ probability of being chosen as a neighbor
  Judge J
  Integer d
  Predicate valid ▷ e.g., valid_full (defined in Algorithm 3)
  Predicate conflict ▷ e.g., $\mathsf{conflict}_{rat}$ (Algorithm 3)
  Assert$((\mathsf{valid}, \mathsf{conflict})$ is propagation-friendly w.r.t. $(J, d))$

Uses
  $\mathcal{N}(\rho^*)$, the random variable that returns a subset of processes in Ψ, where the inclusion of a process follows the Bernoulli distribution with expected value ρ^*.

Local variables:
  Set(Messages) $relayed \leftarrow \emptyset$

upon propagate($c_i \in$ FCertificate):
  send $\langle \text{CERT}, c_i \rangle$ to processes in $\mathcal{N}_{new}$, with $\mathcal{N}_{new} \xleftarrow{\$} \mathcal{N}(\rho^*)$

upon $m_j = \langle \text{CERT}, c_j \rangle$ is received from process p_j:
  if $\mathsf{valid}(c_j) = \mathit{true}$:
    if $certificates_i = \emptyset$:
      $certificates_i \leftarrow certificates_i \cup \{c_j\}$
      send $\langle \text{CERT}, c_j \rangle$ to processes in $\mathcal{N}_{new}$, with $\mathcal{N}_{new} \xleftarrow{\$} \mathcal{N}(\rho^*)$
    if $\exists c_k \in certificates_i$ s.t. $\mathsf{conflict}(c_k, c_j)$:
      broadcast $\langle \text{CONFLICT}, (c_k, c_j) \rangle$

upon the reception of $\langle \text{CONFLICT}, \pi \rangle$ s.t. $\pi \in$ Proof (π is a valid proof of misbehavior)
  trigger generate_proof(π)

Definition 2. Let $x \in \{1, 2\}$. A *x-scalable parametrization* is a tuple $(n, \lambda, t, t_{acc}, \gamma, \epsilon, \delta, \hat{\delta}, \rho_x) \in \mathbb{N}^4 \times \mathbb{R}^5$, s.t. $t = \lceil n(\frac{1}{3} + \epsilon) \rceil - 1$, $t_{acc} = n - \Theta(n)$, $\gamma = (n - t_{acc})/n$, $\epsilon, \delta, \hat{\delta} \in \Omega(1)$, $\rho_1 = \lambda/(\gamma n)$, $\rho_2 = \sqrt{\rho_1}$.

Let $x \in \{1, 2\}$. Let $\mathsf{para} = (n, \lambda, t, t_{acc}, \gamma, \epsilon, \delta, \hat{\delta}, \rho_x)$ be a x-scalable parametrization. We say that the tuple $(\mathsf{mc}_x, \mathsf{rc}_x, d)$, representing message complexity, round detection complexity, and number of dishonest processes exposed in case of disagreement, is a *x-scalable*

tuple w.r.t. para if $d = B^{\epsilon,\lambda}_{\delta,\hat{\delta}} = \frac{\lambda}{3}\big(1 - 3\hat{\delta} - 2(\delta(2+\epsilon) - \epsilon)\big)$, and:

- if $x = 1$, $\mathsf{mc}_1 = O\big(\lambda n/\gamma\big)$, and $\mathsf{rc}_1 = O(\log(\gamma n/\lambda))$,
- if $x = 2$, $\mathsf{mc}_2 = O\big(n^{3/2} \cdot \sqrt{\lambda/\gamma}\big)$, and $\mathsf{rc}_2 = 2$

Theorem 3. Let $x \in \{1, 2\}$. Let $\mathsf{para} = (n, \lambda, t, t_{acc}, \gamma, \epsilon, \delta, \hat{\delta}, \rho_x)$ be a x-scalable parametrization. Let both the Propagator (Module 3) and $\Pi^{prop}_{\mathsf{ERFlood}}\langle\rho^*\rangle$ (Algorithm 4) be parameterized by a pair of predicates $(\mathsf{valid}, \mathsf{conflict})$ that is propagation-friendly with respect to some judge-integer pair (J, d). Then, $\Pi^{prop}_{\mathsf{ERFlood}}\langle\rho^*\rangle$ provides (t_{acc}, d, J)-accountability for the propagator's conflict-freeness property, against a 1-delayed-adaptive adversary. This guarantee holds with expected round complexity rc_x, and expected communication complexity mc_x, such that $(\mathsf{mc}_x, \mathsf{rc}_x, d)$ is a x-scalable tuple w.r.t. para, where each message contains a certificate c verifying $\mathsf{valid}(c)$.

PROOF SKETCH. Define the predicate $\mathsf{conflict\&valid}(c_1, c_2)$ to be *true* if and only if $\mathsf{conflict}(c_1, c_2) = \mathit{true}$ and $\mathsf{valid}(c_1) = \mathsf{valid}(c_2) = \mathit{true}$.

Assume two distinct correct processes p_i and p_j propagate certificates c_1 and c_2 respectively, such that $\mathsf{conflict\&valid}(c_1, c_2) = \mathit{true}$. Let C denote the condition that *no* so-far correct process ever passes the check at line 19. Suppose C does not hold for some so-far correct process p. Then p will broadcast a message of the form $\langle \text{CONFLICT}, \tilde{c}, \tilde{c}'\rangle$ with $\mathsf{conflict\&valid}(\tilde{c}, \tilde{c}') = \mathit{true}$. Since $(\mathsf{valid}, \mathsf{conflict})$ is propagation-friendly with respect to (J, d), each correct process will invoke $\mathsf{generate\_proof}(\tilde{c}, \tilde{c}')$ such that $J(\tilde{c}, \tilde{c}')$ returns a set G of size at least d, except with negligible probability.

Now assume, for contradiction, that C holds forever. Since $(\mathsf{valid}, \mathsf{conflict})$ is propagation-friendly, the predicate $\mathsf{conflict}(\cdot, \cdot)$ induces an equivalence relation $\sim$ over certificates: being non-conflicting is symmetric, reflexive, and transitive. As a result, at any given time, the set of correct processes can be partitioned into four disjoint groups:

- G^0: processes whose *certificates* variable is empty;
- G^1: processes storing a certificate equivalent (i.e., non-conflicting) to c_1;
- G^2: processes storing a certificate equivalent to c_2;
- G^3: processes storing a certificate conflicting with both c_1 and c_2.

Certificates equivalent to c_1 and c_2 then propagate independently, as if under two separate instances of Π_{ERFlood}. When a process in G^0 receives c_1 (resp., c_2), it joins G^1 (resp., G^2). If a process in $G^2 \cup G^3$ (resp., $G^1 \cup G^3$) receives c_1 (resp., c_2), then the conflict predicate conflict triggers at line 19, and the condition C no longer holds. Moreover, if a process in G^1 (resp., G^2) receives a certificate equivalent to c_1 (resp., c_2), it has already relayed an equivalent certificate—faithfully mirroring the propagation behavior of Π_{ERFlood}.

By [15, Theorem 3], a certificate $c'_1 \sim c_1$ (resp., $c'_2 \sim c_2$) will eventually reach a so-far correct process $p \in G^2 \cup G^3$ (resp., $G^1 \cup G^3$). When this occurs, p will pass the check at line 19, contradicting the assumption that C holds forever. Hence, C does not hold indefinitely, and accountability is ensured. To avoid quadratic communication even when exposing malicious processes, the same principle applies: a proof of culpability is never relayed more than once. We omit further details for conciseness.

Now consider the case where no detection occurs (precondition of accountability property is not met). Then no correct process has received conflicting certificates. In that case, due to the check at line 17, each process forwards any given certificate at most once. Thus, the execution mirrors a run of Π_{ERFlood} for a single message, even though different (but non-conflicting) messages may be propagated and the causal structure may differ. Therefore, both the message and communication complexities match those of Π_{ERFlood}, and the latency complexity is no worse. The complexity bounds follow directly from [15, Theorem 3]. □

6. Putting everything together with $\mathcal{ABC}^{++}$

Now, we can sequentially compose the ratifier and the propagator with any (subquadratic) BA protocol, to obtain a (subquadratic) BA protocol with additional forensic support and accountability for agreement. This is specified in Algorithm 5.

Algorithm 5 $\mathcal{ABC}^{++}\langle\rho^*\rangle$ - Pseudocode (for process p_i)

1: **Uses:**
2: Byzantine Agreement, **instance** Π_{BA}
3: Ratifier with $(n, 2, B^{\epsilon,\lambda}_{\delta,\hat{\delta}}, J_{rat}, \mathsf{valid\_full})$ forensic support for agreement, **instance** Π_{rat}
4: Propagator with $(t_{acc}, B^{\epsilon,\lambda}_{\delta,\hat{\delta}}, J_{rat})$ accountability for conflict-freeness, **instance** $(\Pi^{prop}_{\mathsf{ERFlood}}\langle\rho^*, \mathsf{valid\_full}, \mathsf{conflict}_{rat}\rangle)$

5: **upon** $\mathsf{propose}(v_{in} \in \mathsf{Value})$:
6: $\Pi_{BA}.\mathsf{propose}(v_{in})$

7: **upon** $\Pi_{BA}.\mathsf{decide}(v_{predecision} \in \mathsf{Value})$:
8: $\Pi_{rat}.\mathsf{submit}(v_{predecision})$

9: **upon** $\Pi_{rat}.\mathsf{confirm}(v_o \in \mathsf{Value})$:
10: **trigger** $\mathsf{decide}(v_o)$ ▷ achieves BA

11: **upon** $\Pi_{rat}.\mathsf{yield\_certificate}(cert \in \mathsf{FCertificate})$:
12: **trigger** $\mathsf{yield\_certificate}(cert)$ ▷ achieves forensic support

13: $\Pi^{prop}_{\mathsf{ERFlood}}.\mathsf{input}(cert)$ ▷ can be skipped when requiring only forensic support [5]

14: **upon** $\Pi^{prop}_{\mathsf{ERFlood}}.\mathsf{generate\_proof}(\pi \in \mathsf{Proof})$: ▷ achieves accountability
15: **trigger** $\mathsf{generate\_proof}(\pi \in \mathsf{Proof})$ ▷ $J_{rat}(\pi)$ returns a verdict against $B^{\epsilon,\lambda}_{\delta,\hat{\delta}}$ malicious processes

Theorem 4. Let $x \in \{1, 2\}$. Let $\mathsf{para} = (n, \kappa, t, t_{acc}, \gamma, \epsilon, \delta, \hat{\delta}, \rho_x)$ be a x-scalable parametrization, and $(\mathsf{mc}'_x, \mathsf{drc}_x, d)$ be a x-scalable tuple w.r.t. para. Let Π_{BA} be a protocol that solves Byzantine agreement with validity property val, probability $\rho(f)$ of success under f adaptive corruptions, expected round complexity rc, and expected communication complexity cc. Let

$\bar{\Pi}_{BA}$ be a protocol obtained by applying $\mathcal{ABC}^{++}\langle\rho_x\rangle$ (Algorithm 5) to Π_{BA}. The judge J_{rat} (Algorithm 3) is a fair judge for $\bar{\Pi}_{BA}$, which:

- provides $(n, 2, B^{\epsilon,\lambda}_{\delta,\hat{\delta}}, J_{rat}, \mathsf{valid\_full})$-forensic support for the agreement property;
- provides $(t_{acc}, B^{\epsilon,\lambda}_{\delta,\hat{\delta}}, J_{rat})$-accountability against a 1-delayed adaptive adversary, with detection round complexity drc_x;
- solves BA with validity property val, $\bar{\mathsf{rc}} = \mathsf{rc}+1$ round complexity (for decision), $\bar{\mathsf{cc}} = \mathsf{cc} + \mathsf{mc}'_x \cdot \mathsf{ms}'$ and probability $\bar{\rho}(f)$ of success under f adaptive corruptions, where $\bar{\rho}(f) = \rho(f)(1 - neg(\lambda))$ if $f \leq t$ (and 0 otherwise). Here ms' is the size of full certificates.

PROOF. Let f be the actual number of corruptions. First, let us prove the properties of $\bar{\Pi}_{BA}$. Protocol Π_{BA} guarantees that all correct processes submit the same valid value v to Π_{rat} with probability $\rho(f)$ at communication cost cc and round complexity rc. By Theorem 2, all correct processes confirm v from Π_{rat} with probability $\bar{\rho}(f) = (1 - neg(\lambda))\rho(f)$. This means that all correct processes decide the valid value v from $\bar{\Pi}_{BA}$ with probability $\bar{\rho}$, with an additional round and $O(\lambda\kappa n)$ expected communication. The complexity is obtained from Theorem 2 and Theorem 3. Second, fairness of J_{rat} follows directly from the fact that no correct process signs conflicting submit messages. Third, let us prove the accountability and the forensic support property. Assume agreement is violated, i.e., there exist two correct processes p_i and p_j that have decided values v_i and v_j respectively, such that $v_i \neq v_j$. This implies that p_i and p_j have confirmed values v_i and v_j respectively from Π_{rat}. Thus, we can apply Theorem 2, which implies (i) the forensic property and (ii) that p_i and p_j have triggered $\mathsf{propagate}$ to $\Pi^{prop}_{\mathsf{ERFlood}}$ with conflicting certificates. Thus, we can apply Theorem 3, which completes the proof.

□

7. Generalization in the AUC framework

The $\mathcal{ABC}^{++}$ transformation actually extends to any (ideally subquadratic) protocol that implements what [11] refers to as an *easily-accountable agreement task*. This class includes, for example, Byzantine Reliable Broadcast, and Consistent Broadcast (Reliable Broadcast without the totality property) as used in recent BFT protocols that decouple block transmission from block ordering [14].

We formalize this generalization in the Accountable Universally Composable (AUC) framework [18]. Specifically, we show that applying the $\mathcal{ABC}^{++}$ compiler to any easily-accountable agreement functionality $\mathcal{F}$ yields a protocol that AUC-realizes its accountable counterpart $\mathcal{F}^{\text{acc}}$. While the result is stated here, the full proof is deferred to §B.

Definition 3. Let $\mathcal{F}$ be an ideal functionality providing the following interface, where each request or notification may be triggered at most once:

- *Request:* $(\mathsf{propose}, \mathsf{sid}, \mathsf{ssid}, v)$
- *Notification:* $(\mathsf{decide}, \mathsf{sid}, \mathsf{ssid}, w)$

Here, sid, ssid denote the typical (sub-)session IDs used in the UC framework. We say that $\mathcal{F}$ is an *easily accountable agreement* functionality if it satisfies the following properties:

- **Leakage:** All proposals (excluding possibly the payload) are leaked to the adversary. Moreover, any value w decided by a correct process must first be exposed on the adversarial tape and scheduled by the adversary.
- **Agreement:** No two correct processes decide on different values.
- **Termination:** Either of the following holds:
  - **Totality:** If a correct process decides, then all correct processes eventually decide.
  - **Partial-Decidability:** If it is permissible for a correct process p not to decide under some environment behavior β, then it must also be permissible—under the same behavior β—for any subset of correct processes not to decide.

We define $\mathcal{F}^{\text{acc}}$ to be the (m, d, J)-*accountable counterpart* of $\mathcal{F}$ if, in addition to the above interface and guarantees, $\mathcal{F}^{\text{acc}}$ exposes a detection interface of the form:

$$(\mathsf{detection}, \mathsf{sid}, \mathsf{ssid}, D_i, \pi_i),$$

where D_i is a set of process identifiers and π_i is a proof. The functionality guarantees that if agreement is violated under fewer than m corruptions, then every correct process p_i eventually triggers such a detection event, and the (fair and public) [18] judge $J(\pi_i)$ returns the verdict $\mathsf{dis}(D_i)$ with $|D_i| \geq d$.

Theorem 5. Let J_{rat} be the judge defined in Algorithm 3, and $\mathcal{F}^{\text{acc}}_{\text{ea}}$ be the $(t_{acc}, B^{\epsilon,\lambda}_{\delta,\hat{\delta}}, J_{rat})$-accountable counterpart of an easily accountable agreement functionality $\mathcal{F}_{\text{ea}}$ (as defined in Definition 3). If $\mathcal{P}$ UC-realizes $\mathcal{F}_{\text{ea}}$, then $\mathcal{ABC}^{++}(\mathcal{P})$ UC-realizes $\mathcal{F}^{\text{acc}}_{\text{ea}}$ with complexity overhead described in Theorem 4.

8. Evaluation

This section evaluates the practical feasibility of our protocol by examining the constants and cryptographic costs that influence performance. Our focus is on the ratifier's liveness and forensic guarantees, as well as the efficiency of quorum certificate aggregation and verification.

Parameters and their role. We denote by λ the expected committee size, and by $\epsilon \in [0, 1/3]$ the "resilience slack", meaning that the tolerated fault ratio satisfies $t < (\frac{1}{3} - \epsilon)n$.

For a specific instance of the ratifier, let Z_F and Z_L denote the number of elected processes and the number of elected correct processes, respectively. By construction, $\mathbb{E}(Z_F) = \lambda$ and $\mathbb{E}(Z_L) \geq (\frac{2}{3} + \epsilon)\lambda$.

We recall the size of a VRF quorum:

$$W^{\epsilon,\lambda}_{\delta} = (1 - \delta)\left(\tfrac{2}{3} + \epsilon\right)\lambda,$$

where δ captures the allowed deviation from the expected value $\mathbb{E}(Z_L)$ under the corresponding Chernoff bound. If ρ_L denotes the target upper bound on the probability of liveness violation, we require:

$$\rho_L \leq \Pr[Z_L < W_\delta^{\epsilon,\lambda}] = \Pr[Z_L < (1-\delta)\mathbb{E}(Z_L)], \quad (1)$$

which implies:

$$W_\delta^{\epsilon,\lambda} \geq \frac{1-\delta}{\delta^2} \cdot 2\ln\left(\frac{1}{\rho_L}\right). \quad (2)$$

Similarly, we recall the conservative lower bound on the intersection size of two VRF quorums Q and Q':

$$B_{\delta,\hat{\delta}}^{\epsilon,\lambda} = 2W_\delta^{\epsilon,\lambda} - (1+\hat{\delta})\lambda,$$

where $\hat{\delta}$ quantifies the deviation in a Chernoff bound for Z_F. Since

$$|Q \cap Q'| \geq |Q| + |Q'| - |Q \cup Q'| \geq 2W_\delta^{\epsilon,\lambda} - Z_F,$$

we get:

$$\Pr[|Q \cap Q'| \leq B_{\delta,\hat{\delta}}] = \Pr[Z_F \geq (1+\hat{\delta})\mathbb{E}(Z_F)], \quad (3)$$

which implies:

$$\lambda \geq \frac{2+\hat{\delta}}{\hat{\delta}^2} \cdot \ln\left(\frac{1}{\rho_F}\right), \quad (4)$$

where ρ_F denotes the upper bound on the probability that the intersection is too small.

Larger values of δ and $\hat{\delta}$ tighten the Chernoff bounds but reduce the guaranteed overlap size $B_{\delta,\hat{\delta}}^{\epsilon,\lambda}$ between quorums. Therefore, we must ensure that

$$B_{\delta,\hat{\delta}}^{\epsilon,\lambda} = \left(2(1-\delta)\left(\frac{2}{3}+\epsilon\right) - (1+\hat{\delta})\right)\lambda \geq d \quad (5)$$

for $d \geq 1$ (or even 100).

and ideally, we want this overlap to be substantially large, e.g., $B_{\delta,\hat{\delta}}^{\epsilon,\lambda} \geq 100$, which imposes constraints on the tuple $(\epsilon, \delta, \hat{\delta}, \lambda)$.

Assume we want to bound the probability of a security violation by $\rho_F = \rho_L = 10^{-12} \simeq 2^{-40}$, under the assumption that 80% of the processes are correct in the optimistic case (i.e., $\frac{2}{3} + \epsilon = 0.8$), and that at most $W_{\delta,\hat{\delta}}^{\epsilon,\lambda} = 1000$ signatures are aggregated during ratification.

A suitable choice of parameters is:

$$\begin{cases} \epsilon = 2/15 = 0.8 - 2/3 \\ \delta = 0.21 \text{ (to satisfy (2))} \\ \lambda = 1582 \text{ by the definition of } W_\delta^{\epsilon,\lambda} \\ \hat{\delta} = 0.2 \text{ (to satisfy (4))} \\ \lfloor B_{\delta,\hat{\delta}}^{\epsilon,\lambda} \rfloor = 101 \text{ by the definition of } B_{\delta,\hat{\delta}}^{\epsilon,\lambda} \end{cases}$$

We do not elaborate on the propagation guarantees, as they directly follow from Theorem 3 of [15]. For instance, setting $\lambda \geq 1500$ and assuming a degraded mode with $\gamma = 0.01$ (i.e., 1% of processes remain correct) yields a probability p_{bad} of security violation several orders of magnitude below $2^{-40} \approx 10^{-12}$, even for a system size as large as $n = 10^6$ or $n = 10^9$. For comparison, the current number of Ethereum validators is around $n = 10^6$. In fact, the probability p_{bad} does not need to be extremely low to act as a deterrent: to break accountability, the adversary must commit to delivering two conflicting certificates to two different processes, without knowing whether the second certificate will avoid reaching the process that already received the first—thus risking detection. This could allow the use of a smaller statistical security parameter $\lambda' < \lambda$ for propagation, further reducing communication overhead.

VRF Quorum Certificates. The exact complexity of the accountable confirmer essentially reduces to the computation, verification, and propagation of VRF-quorum certificates built from W committee members.

Each SUBMIT message includes a hash value, a VRF proof, and a multi-signature. Both the VRF scheme and the multi-signature scheme can be instantiated using the BLS signature scheme [95]. As a result, computing a SUBMIT message is fast: it requires only two BLS signatures and two hash evaluations (one for the message value and one for the eligibility check). The size of the message is also compact: it consists of two BLS signatures and one hash.

During aggregation, for each SUBMIT message of the form

$$\langle \text{SUBMIT}, h, \sigma_i^{msig}, \sigma_i^{vrf} \rangle,$$

a process must:

1) Hash the BLS signature σ_i^{vrf} to check the eligibility of p_i,
2) Verify the VRF signature σ_i^{vrf},
3) Verify the multi-signature σ_i^{msig},
4) Aggregate:
   (a) the BLS signature σ_i^{vrf},
   (b) the BLS signature σ_i^{msig},
   with the corresponding partial aggregations under construction.

However, verifying BLS signatures is computationally expensive due to the required pairing operations. This overhead becomes significant in what we refer to as the *pessimistic aggregation mode*.

Optimizations. To mitigate this cost, we propose an optimization. Instead of immediately verifying each signature, the aggregator can:

(4') Perform step (4) as above, while also aggregating:
   (c) the public keys of the signers.

Then, instead of performing $2W$ verifications, the aggregator performs only:

(2') A single verification of the final aggregated VRF signature,
(3') A single verification of the final aggregated multi-signature.

This approach is what we call the *super-optimistic aggregation mode*. However, a single malformed signature

can invalidate the final result, rendering the optimization ineffective.

To address this, we introduce the *optimistic aggregation mode*. In this setting, each signer additionally signs its SUBMIT message (which includes the two BLS signatures) using a lightweight, non-aggregated signature scheme such as EdDSA. The aggregator:

- Aggregates as in the super-optimistic mode,
- Verifies only the efficient signature before accepting the message.

If f' faulty processes attempt to corrupt the aggregation with malformed messages, they will be caught via their efficient signatures. In blockchain-based systems (e.g., based on Proof-of-Stake), this can be coupled with deposit slashing via proof-of-misbehavior, creating a strong economic disincentive against such attacks. Furthermore, by storing aggregated signatures and public keys in a tree-like structure, the aggregator can identify the faulty signers in $O(f' \log W)$ BLS verifications.

Finally, a verifier in the propagator component can apply the same verification technique as the aggregator, reusing the aggregation structure to validate the quorum certificate efficiently.

Evaluation. We evaluate the cost of handling VRF-based quorum certificates—including aggregation, verification, and communication—through a Rust implementation, using the Criterion benchmark crate [96], and executed on a Macbook Pro (2021) with an Apple M1 processor, 16 GB of RAM, and MacOS Sequoia 15.0.1. Our implementation uses the `blstrs` library [97] with the BLS12-381 curve [98], which provides efficient finite field and elliptic curve operations for BLS signatures. For non-aggregatable signatures used in the optimistic aggregation mode, we rely on the `ed25519_dalek` library [99]. [4]

When instantiated with a quorum size of $W = 1000$ signers:

- The computation of a SUBMIT message takes less than **0.5ms**,
- The verification of a full quorum certificate takes approximately **49ms**. Importantly, this verification occurs only once per decision in the nominal mode, and twice in case of (detected) safety violation.
- The aggregation time varies depending on the mode:
  - **Pessimistic mode:** 647ms,
  - **Optimistic mode:** 45ms,
  - **Super-Optimistic mode:** 11ms.

A quorum certificate must include a proof of eligibility. Without it, in the event of a disagreement, the adversary could fabricate conflicting certificates using non-elected signers whose intersection is empty, while withholding honest certificates to save bandwidth.

Including individual eligibility proofs prevents this issue but increases the certificate size by a factor of W. Importantly, this does not impact verification time, as explained earlier, but it does raise bandwidth concerns. For instance, with $W = 1000$, the W BLS signatures for eligibility account for roughly **52KB**, which is approximately **10%** of a typical Bitcoin block size, i.e. still reasonable in many deployment contexts. By slightly relaxing the adversarial model to tolerate a 3-delayed adaptive adversary, certificate dissemination can be optimized into a three-step process: (1) first, the sender transmits only the hash of the certificate; (2) upon receiving an explicit request, (3) the sender delivers the full certificate. This approach ensures that in the nominal case, each correct process receives the full certificate for a given hash value at most once.

Alternatively, eligibility proofs can be combined into a single succinct non-interactive argument, which would be significantly smaller. In particular, given public knowledge of a light certificate $c = (s, h, G, \sigma)$ and the public keys $\{pk_i\}_{p_i \in G}$, the prover wants to prove, in addition to $\mathsf{light\_valid}(c)$, that $G \subseteq C(s, \lambda)$, i.e., that for all $p_i \in G$, there exists σ_i such that $\mathsf{committee\_val}(s, \lambda, i, \sigma_i) = true$. Such a proof could be of size $O(\kappa + W \log n)$ by encoding the W public keys as indices (since processes can query these from the PKI). Instead of using a generic STARK solution, one can use Jackpot [17], a recent and concretely efficient construction for non-interactive aggregatable lotteries. Jackpot offers several advantages in our setting: it is proven secure in the UC framework; the aggregate proof consists of only two group elements (approximately 80 bytes when instantiated over BLS12-381); and it supports fast aggregation and verification, with measured runtimes below 7ms and 9ms respectively for 1000 signers on very similar hardware, which aligns with our own benchmarks in the super-optimistic mode. On the other hand, the construction relies on the algebraic group model, assumes a q-SDH setup, and requires participants to publish fresh public keys via the bulletin-board PKI every q lotteries.

9. Conclusion

In this work, we introduced $\mathcal{ABC}^{++}$, a generic transformation that enhances BA protocols with accountability. By leveraging two key primitives, the ratifier and the propagator, we achieve delayed-adaptively-secure accountability with subquadratic communication complexity. This transformation is applicable to a wide range of (subquadratic) BA and Byzantine Reliable Broadcast protocols, offering the first subquadratic accountable counterparts of these primitives. Such a subquadratic accountable Byzantine Reliable broadcast can be then plugged into the transformation τ_{scr} of [12], to obtain τ_{scr}^{++}, which transforms any deterministic (and even beyond) distributed protocol into its accountable counterpart, with a subquadratic multiplicative communication overhead only.

We conjecture that the aforementioned transformations can be easily adapted to Proof-of-Stake blockchains with weighted validators, using weighted VRFs [17], [100] and weighted flooding [44], [45].

4. our code is available at https://github.com/pcivit/abcpp

References

[1] G. Bracha and S. Toueg, "Resilient consensus protocols," in *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing, Montreal, Quebec, Canada, August 17-19, 1983*, R. L. Probert, N. A. Lynch, and N. Santoro, Eds. ACM, 1983, pp. 12–26. [Online]. Available: https://doi.org/10.1145/800221.806706

[2] P. Civit, S. Gilbert, V. Gramoli, R. Guerraoui, and J. Komatovic, "As easy as ABC: optimal (a)ccountable (b)yzantine (c)onsensus is easy!" *J. Parallel Distributed Comput.*, vol. 181, p. 104743, 2023. [Online]. Available: https://doi.org/10.1016/j.jpdc.2023.104743

[3] A. Momose and L. Ren, "Multi-threshold byzantine fault tolerance," in *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, Y. Kim, J. Kim, G. Vigna, and E. Shi, Eds. ACM, 2021, pp. 1686–1699. [Online]. Available: https://doi.org/10.1145/3460120.3484554

[4] P. Sheng, G. Wang, K. Nayak, S. Kannan, and P. Viswanath, "BFT protocol forensics," in *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, Y. Kim, J. Kim, G. Vigna, and E. Shi, Eds. ACM, 2021, pp. 1722–1743. [Online]. Available: https://doi.org/10.1145/3460120.3484566

[5] ——, "Player-replaceability and forensic support are two sides of the same (crypto) coin," in *Financial Cryptography and Data Security - 27th International Conference, FC 2023, Bol, Brač, Croatia, May 1-5, 2023, Revised Selected Papers, Part I*, ser. Lecture Notes in Computer Science, F. Baldimtsi and C. Cachin, Eds., vol. 13950. Springer, 2023, pp. 56–74. [Online]. Available: https://doi.org/10.1007/978-3-031-47754-6_4

[6] W. Tang, P. Sheng, R. Ni, P. Roy, X. Wang, G. Fanti, and P. Viswanath, "Cft-forensics: High-performance byzantine accountability for crash fault tolerant protocols," *arXiv preprint arXiv:2305.09123*, 2023.

[7] Q. You, H. Yang, X. Zhang, X. Jiang, K. Guo, and K. Hu, "Forensic support for abraham et al.'s bb protocol," *Entropy*, vol. 27, no. 5, p. 504, 2025.

[8] R. Küsters, T. Truderung, and A. Vogt, "Accountability: definition and relationship to verifiability," in *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010*, E. Al-Shaer, A. D. Keromytis, and V. Shmatikov, Eds. ACM, 2010, pp. 526–535. [Online]. Available: https://doi.org/10.1145/1866307.1866366

[9] L. F. de Souza, P. Kuznetsov, T. Rieutord, and S. Tucci Piergiovanni, "Accountability and reconfiguration: Self-healing lattice agreement," in *25th International Conference on Principles of Distributed Systems, OPODIS 2021, December 13-15, 2021, Strasbourg, France*, ser. LIPIcs, Q. Bramas, V. Gramoli, and A. Milani, Eds., vol. 217. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, pp. 25:1–25:23. [Online]. Available: https://doi.org/10.4230/LIPIcs.OPODIS.2021.25

[10] P. Civit, S. Gilbert, and V. Gramoli, "Polygraph: Accountable byzantine agreement," in *41st IEEE International Conference on Distributed Computing Systems, ICDCS 2021, Washington DC, USA, July 7-10, 2021*. IEEE, 2021, pp. 403–413. [Online]. Available: https://doi.org/10.1109/ICDCS51616.2021.00046

[11] P. Civit, S. Gilbert, V. Gramoli, R. Guerraoui, and J. Komatovic, "As easy as ABC: optimal (a)ccountable (b)yzantine (c)onsensus is easy!" in *2022 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2022, Lyon, France, May 30 - June 3, 2022*. IEEE, 2022, pp. 560–570. [Online]. Available: https://doi.org/10.1109/IPDPS53621.2022.00061

[12] P. Civit, S. Gilbert, V. Gramoli, R. Guerraoui, J. Komatovic, Z. Milosevic, and A. Seredinschi, "Crime and Punishment in Distributed Byzantine Decision Tasks," in *42nd IEEE International Conference on Distributed Computing Systems, ICDCS 2022, Bologna, Italy, July 10-13, 2022*. IEEE, 2022, pp. 34–44. [Online]. Available: https://doi.org/10.1109/ICDCS54860.2022.00013

[13] M. Yin, D. Malkhi, M. K. Reiter, G. Golan-Gueta, and I. Abraham, "Hotstuff: BFT consensus with linearity and responsiveness," in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, P. Robinson and F. Ellen, Eds. ACM, 2019, pp. 347–356. [Online]. Available: https://doi.org/10.1145/3293611.3331591

[14] X. Wang, H. Wang, H. Zhang, and S. Duan, "Pando: Extremely scalable BFT based on committee sampling," *IACR Cryptol. ePrint Arch.*, p. 664, 2024. [Online]. Available: https://eprint.iacr.org/2024/664

[15] C. Matt, J. B. Nielsen, and S. E. Thomsen, "Formalizing delayed adaptive corruptions and the security of flooding networks," in *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part II*, ser. Lecture Notes in Computer Science, Y. Dodis and T. Shrimpton, Eds., vol. 13508. Springer, 2022, pp. 400–430. [Online]. Available: https://doi.org/10.1007/978-3-031-15979-4_14

[16] S. Micali, M. O. Rabin, and S. P. Vadhan, "Verifiable random functions," in *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*. IEEE Computer Society, 1999, pp. 120–130. [Online]. Available: https://doi.org/10.1109/SFFCS.1999.814584

[17] N. Fleischhacker, M. Hall-Andersen, M. Simkin, and B. Wagner, "Jackpot: Non-interactive aggregatable lotteries," in *Advances in Cryptology - ASIACRYPT 2024 - 30th International Conference on the Theory and Application of Cryptology and Information Security, Kolkata, India, December 9-13, 2024, Proceedings, Part VI*, ser. Lecture Notes in Computer Science, K. Chung and Y. Sasaki, Eds., vol. 15489. Springer, 2024, pp. 365–397. [Online]. Available: https://doi.org/10.1007/978-981-96-0938-3_12

[18] M. Graf, R. Küsters, and D. Rausch, "AUC: accountable universal composability," in *44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023*. IEEE, 2023, pp. 1148–1167. [Online]. Available: https://doi.org/10.1109/SP46215.2023.10179384

[19] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai, "Universally composable two-party and multi-party secure computation," in *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, J. H. Reif, Ed. ACM, 2002, pp. 494–503. [Online]. Available: https://doi.org/10.1145/509907.509980

[20] R. Cohen, "Asynchronous secure multiparty computation in constant time," in *Public-Key Cryptography - PKC 2016 - 19th IACR International Conference on Practice and Theory in Public-Key Cryptography, Taipei, Taiwan, March 6-9, 2016, Proceedings, Part II*, ser. Lecture Notes in Computer Science, C. Cheng, K. Chung, G. Persiano, and B. Yang, Eds., vol. 9615. Springer, 2016, pp. 183–207. [Online]. Available: https://doi.org/10.1007/978-3-662-49387-8_8

[21] M. Rambaud and A. Urban, "Almost-asynchronous MPC under honest majority, revisited," *IACR Cryptol. ePrint Arch.*, p. 503, 2021. [Online]. Available: https://eprint.iacr.org/2021/503

[22] T. Gong, G. F. Camilo, K. Nayak, A. Lewis-Pye, and A. Kate, "Recover from excessive faults in partially-synchronous BFT SMR," *IACR Cryptol. ePrint Arch.*, p. 83, 2025. [Online]. Available: https://eprint.iacr.org/2025/083

[23] A. F. Anta, K. M. Konwar, C. Georgiou, and N. C. Nicolaou, "Formalizing and implementing distributed ledger objects," *SIGACT News*, vol. 49, no. 2, pp. 58–76, 2018. [Online]. Available: https://doi.org/10.1145/3232679.3232691

[24] J. A. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," in *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, ser. Lecture Notes in Computer Science, E. Oswald and M. Fischlin, Eds., vol. 9057. Springer, 2015, pp. 281–310. [Online]. Available: https://doi.org/10.1007/978-3-662-46803-6_10

[25] ——, "The bitcoin backbone protocol: Analysis and applications," *J. ACM*, vol. 71, no. 4, pp. 25:1–25:49, 2024. [Online]. Available: https://doi.org/10.1145/3653445

[26] M. Graf, R. Küsters, D. Rausch, S. Egger, M. Bechtold, and M. Flinspach, "Accountable bulletin boards: Definition and provably secure implementation," *IACR Cryptol. ePrint Arch.*, p. 1869, 2023. [Online]. Available: https://eprint.iacr.org/2023/1869

[27] A. Kiayias, A. Kuldmaa, H. Lipmaa, J. Siim, and T. Zacharias, "On the security properties of e-voting bulletin boards," in *Security and Cryptography for Networks - 11th International Conference, SCN 2018, Amalfi, Italy, September 5-7, 2018, Proceedings*, ser. Lecture Notes in Computer Science, D. Catalano and R. D. Prisco, Eds., vol. 11035. Springer, 2018, pp. 505–523. [Online]. Available: https://doi.org/10.1007/978-3-319-98113-0_27

[28] M. Camaioni, R. Guerraoui, J. Komatovic, M. Monti, P. Roman, M. Vidigueira, and G. Voron, "Carbon: Scaling trusted payments with untrusted machines," *IEEE Trans. Dependable Secur. Comput.*, vol. 22, no. 2, pp. 1168–1180, 2025. [Online]. Available: https://doi.org/10.1109/TDSC.2024.3428617

[29] D. Collins, R. Guerraoui, J. Komatovic, P. Kuznetsov, M. Monti, M. Pavlovic, Y. Pignolet, D. Seredinschi, A. Tonkikh, and A. Xygkis, "Online payments by merely broadcasting messages," in *50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2020, Valencia, Spain, June 29 - July 2, 2020*. IEEE, 2020, pp. 26–38. [Online]. Available: https://doi.org/10.1109/DSN48063.2020.00023

[30] R. Guerraoui, P. Kuznetsov, M. Monti, M. Pavlovic, and D. Seredinschi, "The consensus number of a cryptocurrency," in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, P. Robinson and F. Ellen, Eds. ACM, 2019, pp. 307–316. [Online]. Available: https://doi.org/10.1145/3293611.3331589

[31] A. Ranchal-Pedrosa and V. Gramoli, "TRAP: the bait of rational players to solve byzantine consensus," in *ASIA CCS '22: ACM Asia Conference on Computer and Communications Security, Nagasaki, Japan, 30 May 2022 - 3 June 2022*, Y. Suga, K. Sakurai, X. Ding, and K. Sako, Eds. ACM, 2022, pp. 168–181. [Online]. Available: https://doi.org/10.1145/3488932.3517386

[32] A. Lewis-Pye and T. Roughgarden, "Beyond optimal fault tolerance," *CoRR*, vol. abs/2501.06044, 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2501.06044

[33] A. Ranchal-Pedrosa and V. Gramoli, "ZLB: A blockchain to tolerate colluding majorities," in *54th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2024, Brisbane, Australia, June 24-27, 2024*. IEEE, 2024, pp. 209–222. [Online]. Available: https://doi.org/10.1109/DSN58291.2024.00032

[34] S. Sridhar, D. Zindros, and D. Tse, "Better safe than sorry: Recovering after adversarial majority," *CoRR*, vol. abs/2310.06338, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2310.06338

[35] A. Haeberlen, P. Kouznetsov, and P. Druschel, "Peerreview: practical accountability for distributed systems," in *Proceedings of the 21st ACM Symposium on Operating Systems Principles 2007, SOSP 2007, Stevenson, Washington, USA, October 14-17, 2007*, T. C. Bressoud and M. F. Kaashoek, Eds. ACM, 2007, pp. 175–188. [Online]. Available: https://doi.org/10.1145/1294261.1294279

[36] R. Künnemann, I. Esiyok, and M. Backes, "Automated verification of accountability in security protocols," in *32nd IEEE Computer Security Foundations Symposium, CSF 2019, Hoboken, NJ, USA, June 25-28, 2019*. IEEE, 2019, pp. 397–413. [Online]. Available: https://doi.org/10.1109/CSF.2019.00034

[37] M. Backes, P. Druschel, A. Haeberlen, and D. Unruh, "CSAR: A practical and provable technique to make randomized systems accountable," in *Proceedings of the Network and Distributed System Security Symposium, NDSS 2009, San Diego, California, USA, 8th February - 11th February 2009*. The Internet Society, 2009.

[38] M. Backes, D. Fiore, and E. Mohammadi, "Privacy-preserving accountable computation," in *Computer Security - ESORICS 2013 - 18th European Symposium on Research in Computer Security, Egham, UK, September 9-13, 2013. Proceedings*, ser. Lecture Notes in Computer Science, J. Crampton, S. Jajodia, and K. Mayes, Eds., vol. 8134. Springer, 2013, pp. 38–56. [Online]. Available: https://doi.org/10.1007/978-3-642-40203-6_3

[39] A. Papadimitriou, M. Zhao, and A. Haeberlen, "Towards privacy-preserving fault detection," in *Proceedings of the 9th Workshop on Hot Topics in Dependable Systems, HotDep 2013, Farmington, Pennsylvania, USA, November 3, 2013*, C. Cachin and R. van Renesse, Eds. ACM, 2013, pp. 6:1–6:5. [Online]. Available: https://doi.org/10.1145/2524224.2524233

[40] J. Chen and S. Micali, "Algorand: A secure and efficient distributed ledger," *Theor. Comput. Sci.*, vol. 777, pp. 155–183, 2019. [Online]. Available: https://doi.org/10.1016/j.tcs.2019.02.001

[41] K. Nayak, "Player Replaceability - Towards Adaptive Security and Sub-quadratic Communication Simultaneously (Part I)," 2023, https://decentralizedthoughts.github.io/2023-01-05-player-replaceability-I/.

[42] ——, "Player Replaceability - Towards Adaptive Security and Sub-quadratic Communication Simultaneously (Part II)," 2023, https://decentralizedthoughts.github.io/2023-01-05-player-replaceability-II/.

[43] A. Kermarrec, L. Massoulié, and A. J. Ganesh, "Probabilistic reliable dissemination in large-scale systems," *IEEE Trans. Parallel Distributed Syst.*, vol. 14, no. 3, pp. 248–258, 2003. [Online]. Available: https://doi.org/10.1109/TPDS.2003.1189583

[44] C. Liu-Zhang, C. Matt, U. Maurer, G. Rito, and S. E. Thomsen, "Practical provably secure flooding for blockchains," in *Advances in Cryptology - ASIACRYPT 2022 - 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5-9, 2022, Proceedings, Part I*, ser. Lecture Notes in Computer Science, S. Agrawal and D. Lin, Eds., vol. 13791. Springer, 2022, pp. 774–805. [Online]. Available: https://doi.org/10.1007/978-3-031-22963-3_26

[45] C. Liu-Zhang, C. Matt, and S. E. Thomsen, "Asymptotically optimal message dissemination with applications to blockchains," in *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part III*, ser. Lecture Notes in Computer Science, M. Joye and G. Leander, Eds., vol. 14653. Springer, 2024, pp. 64–95. [Online]. Available: https://doi.org/10.1007/978-3-031-58734-4_3

[46] V. King and J. Saia, "Breaking the $o(n^2)$ bit barrier: Scalable byzantine agreement with an adaptive adversary," *Journal of the ACM (JACM)*, vol. 58, no. 4, pp. 1–24, 2011.

[47] E. Blum, J. Katz, C. Liu-Zhang, and J. Loss, "Asynchronous byzantine agreement with subquadratic communication," in *Theory of Cryptography - 18th International Conference, TCC 2020, Durham, NC, USA, November 16-19, 2020, Proceedings, Part I*, ser. Lecture Notes in Computer Science, R. Pass and K. Pietrzak, Eds., vol. 12550. Springer, 2020, pp. 353–380. [Online]. Available: https://doi.org/10.1007/978-3-030-64375-1_13

[48] M. Rambaud, "Adaptively secure consensus with linear complexity and constant round under honest majority in the bare PKI model, and separation bounds from the idealized message-authentication model," *IACR Cryptol. ePrint Arch.*, p. 1757, 2023. [Online]. Available: https://eprint.iacr.org/2023/1757

[49] I. Abraham, T. H. Chan, D. Dolev, K. Nayak, R. Pass, L. Ren, and E. Shi, "Communication Complexity of Byzantine Agreement, Revisited," in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, P. Robinson and F. Ellen, Eds. ACM, 2019, pp. 317–326. [Online]. Available: https://doi.org/10.1145/3293611.3331629

[50] I. Abraham, K. Nayak, and N. Shrestha, "Communication and Round Efficient Parallel Broadcast Protocols," *Cryptology ePrint Archive*, 2023.

[51] I. Abraham, E. Chouatt, I. Damgård, Y. Gilad, G. Stern, and S. Yakoubov, "Asynchronous algorand: Reaching agreement with near linear communication and constant expected time," in *Proceedings of the 2025 ACM Symposium on Principles of Distributed Computing, PODC 2025, Huatulco, Mexico, June 16-20, 2025*. ACM, 2025. [Online]. Available: https://eprint.iacr.org/2025/303

[52] S. Cohen, I. Keidar, and A. Spiegelman, "Not a COINcidence: Sub-Quadratic Asynchronous Byzantine Agreement WHP," in *34th International Symposium on Distributed Computing (DISC 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[53] A. Bhangale, C. Liu-Zhang, J. Loss, and K. Nayak, "Efficient adaptively-secure byzantine agreement for long messages," in *Advances in Cryptology - ASIACRYPT 2022 - 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5-9, 2022, Proceedings, Part I*, ser. Lecture Notes in Computer Science, S. Agrawal and D. Lin, Eds., vol. 13791. Springer, 2022, pp. 504–525. [Online]. Available: https://doi.org/10.1007/978-3-031-22963-3_17

[54] G. Bracha, "An Asynchronous [(n-1)/3]-Resilient Consensus Protocol," in *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing, Vancouver, B. C., Canada, August 27-29, 1984*, T. Kameda, J. Misra, J. G. Peters, and N. Santoro, Eds. ACM, 1984, pp. 154–162. [Online]. Available: https://doi.org/10.1145/800222.806743

[55] ——, "Asynchronous byzantine agreement protocols," *Inf. Comput.*, vol. 75, no. 2, pp. 130–143, 1987. [Online]. Available: https://doi.org/10.1016/0890-5401(87)90054-X

[56] R. Guerraoui, P. Kuznetsov, M. Monti, M. Pavlovic, and D. Seredinschi, "Scalable byzantine reliable broadcast," in *33rd International Symposium on Distributed Computing, DISC 2019, October 14-18, 2019, Budapest, Hungary*, ser. LIPIcs, J. Suomela, Ed., vol. 146. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, pp. 22:1–22:16.

[57] Q. Zhao, G. Pîrlea, K. Grzeszkiewicz, S. Gilbert, and I. Sergey, "Compositional verification of composite byzantine protocols," in *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14–18, 2024*. ACM, 2024. [Online]. Available: https://doi.org/10.1145/3658644.3690355

[58] W. Tang, P. Sheng, R. Ni, P. Roy, X. Wang, G. Fanti, and P. Viswanath, "Cft-forensics: High-performance byzantine accountability for crash fault tolerant protocols," in *6th Conference on Advances in Financial Technologies, AFT 2024, September 23-25, 2024, Vienna, Austria*, ser. LIPIcs, R. Böhme and L. Kiffer, Eds., vol. 316. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024, pp. 3:1–3:25. [Online]. Available: https://doi.org/10.4230/LIPIcs.AFT.2024.3

[59] E. Buchman, R. Guerraoui, J. Komatovic, Z. Milosevic, D. Seredinschi, and J. Widder, "Revisiting tendermint: Design tradeoffs, accountability, and practical use," in *52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2022, Supplemental Volume, Baltimore, MD, USA, June 27-30, 2022*. IEEE, 2022, pp. 11–14. [Online]. Available: https://doi.org/10.1109/DSN-S54099.2022.00014

[60] A. D. Pozzo and T. Rieutord, "Fork accountability in tenderbake," in *5th International Symposium on Foundations and Applications of Blockchain 2022, FAB 2022, June 3, 2022, Berkeley, CA, USA*, ser. OASIcs, S. T. Piergiovanni and N. Crooks, Eds., vol. 101. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, pp. 5:1–5:22. [Online]. Available: https://doi.org/10.4230/OASIcs.FAB.2022.5

[61] J. Neu, S. Sridhar, L. Yang, and D. Tse, "Optimal flexible consensus and its application to ethereum," in *IEEE Symposium on Security and Privacy, SP 2024, San Francisco, CA, USA, May 19-23, 2024*. IEEE, 2024, pp. 3885–3903. [Online]. Available: https://doi.org/10.1109/SP54263.2024.00135

[62] P. Civit, S. Gilbert, R. Guerraoui, J. Komatovic, and M. Vidigueira, "On the Validity of Consensus," in *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing, PODC 2023, Orlando, FL, USA, June 19-23, 2023*, R. Oshman, A. Nolin, M. M. Halldórsson, and A. Balliu, Eds. ACM, 2023, pp. 332–343. [Online]. Available: https://doi.org/10.1145/3583668.3594567

[63] H. Cheng, Y. Lu, Z. Lu, Q. Tang, Y. Zhang, and Z. Zhang, "Jumbo: Fully asynchronous bft consensus made truly scalable," *arXiv preprint arXiv:2403.11238*, 2024.

[64] H. Feng, Z. Lu, T. Mai, and Q. Tang, "Making hash-based mvba great again," *Cryptology ePrint Archive*, 2024.

[65] Y. Lu, Z. Lu, and Q. Tang, "Bolt-dumbo transformer: Asynchronous consensus as fast as the pipelined BFT," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, H. Yin, A. Stavrou, C. Cremers, and E. Shi, Eds. ACM, 2022, pp. 2159–2173. [Online]. Available: https://doi.org/10.1145/3548606.3559346

[66] Y. Lu, Z. Lu, Q. Tang, and G. Wang, "Dumbo-MVBA: Optimal Multi-Valued Validated Asynchronous Byzantine Agreement, Revisited," *Proceedings of the Annual ACM Symposium on Principles of Distributed Computing*, pp. 129–138, 2020.

[67] Y. Zhou, Z. Zhang, H. Zhang, S. Duan, B. Hu, L. Wang, and J. Liu, "Dory: Asynchronous BFT with reduced communication and improved efficiency," *IACR Cryptol. ePrint Arch.*, p. 1709, 2022. [Online]. Available: https://eprint.iacr.org/2022/1709

[68] I. Abraham, G. Asharov, A. Patra, and G. Stern, "Perfectly secure asynchronous agreement on a core set in constant expected time," *IACR Cryptol. ePrint Arch.*, p. 1130, 2023. [Online]. Available: https://eprint.iacr.org/2023/1130

[69] R. Cohen, P. Forghani, J. Garay, R. Patel, and V. Zikas, "Concurrent asynchronous byzantine agreement in expected-constant rounds, revisited," in *Theory of Cryptography Conference*. Springer, 2023, pp. 422–451.

[70] S. Das, S. Duan, S. Liu, A. Momose, L. Ren, and V. Shoup, "Asynchronous consensus without trusted setup or public-key cryptography," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024*, B. Luo, X. Liao, J. Xu, E. Kirda, and D. Lie, Eds. ACM, 2024, pp. 3242–3256. [Online]. Available: https://doi.org/10.1145/3658644.3670327

[71] V. Shoup, "A theoretical take on a practical consensus protocol," *IACR Cryptol. ePrint Arch.*, p. 696, 2024. [Online]. Available: https://eprint.iacr.org/2024/696

[72] M. Graf, R. Küsters, and D. Rausch, "Accountability in a permissioned blockchain: Formal analysis of hyperledger fabric," in *IEEE European Symposium on Security and Privacy, EuroS&P 2020, Genoa, Italy, September 7-11, 2020*. IEEE, 2020, pp. 236–255. [Online]. Available: https://doi.org/10.1109/EuroSP48549.2020.00023

[73] J. Neu, E. N. Tas, and D. Tse, "Ebb-and-flow protocols: A resolution of the availability-finality dilemma," in *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*. IEEE, 2021, pp. 446–465. [Online]. Available: https://doi.org/10.1109/SP40001.2021.00045

[74] V. Buterin and V. Griffith, "Casper the friendly finality gadget," *CoRR*, vol. abs/1710.09437, 2017. [Online]. Available: http://arxiv.org/abs/1710.09437

[75] S. Sankagiri, X. Wang, S. Kannan, and P. Viswanath, "Blockchain CAP theorem allows user-dependent adaptivity and finality," in *Financial Cryptography and Data Security - 25th International Conference, FC 2021, Virtual Event, March 1-5, 2021, Revised Selected Papers, Part II*, ser. Lecture Notes in Computer Science, N. Borisov and C. Díaz, Eds., vol. 12675. Springer, 2021, pp. 84–103. [Online]. Available: https://doi.org/10.1007/978-3-662-64331-0_5

[76] J. Neu, E. N. Tas, and D. Tse, "The availability-accountability dilemma and its resolution via accountability gadgets," in *Financial Cryptography and Data Security - 26th International Conference, FC 2022, Grenada, May 2-6, 2022, Revised Selected Papers*, ser. Lecture Notes in Computer Science, I. Eyal and J. A. Garay, Eds., vol. 13411. Springer, 2022, pp. 541–559. [Online]. Available: https://doi.org/10.1007/978-3-031-18283-9_27

[77] ——, "Short paper: Accountable safety implies finality," in *Financial Cryptography and Data Security - 28th International Conference, FC 2024, Curacao Marriott Beach Resort Willemstad, Curaçao, Mars 4-8, 2024, Revised Selected Papers, Part I*, ser. Lecture Notes in Computer Science. Springer, 2024.

[78] R. Cleve, "Limits on the security of coin flips when half the processors are faulty (extended abstract)," in *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA*, J. Hartmanis, Ed. ACM, 1986, pp. 364–369. [Online]. Available: https://doi.org/10.1145/12130.12168

[79] Y. Ishai, R. Ostrovsky, and V. Zikas, "Secure multi-party computation with identifiable abort," in *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II*, ser. Lecture Notes in Computer Science, J. A. Garay and R. Gennaro, Eds., vol. 8617. Springer, 2014, pp. 369–386. [Online]. Available: https://doi.org/10.1007/978-3-662-44381-1_21

[80] R. Cohen and Y. Lindell, "Fairness versus guaranteed output delivery in secure multiparty computation," in *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II*, ser. Lecture Notes in Computer Science, P. Sarkar and T. Iwata, Eds., vol. 8874. Springer, 2014, pp. 466–485. [Online]. Available: https://doi.org/10.1007/978-3-662-45608-8_25

[81] ——, "Fairness versus guaranteed output delivery in secure multiparty computation," *J. Cryptol.*, vol. 30, no. 4, pp. 1157–1186, 2017. [Online]. Available: https://doi.org/10.1007/s00145-016-9245-5

[82] Y. Aumann and Y. Lindell, "Security against covert adversaries: Efficient protocols for realistic adversaries," in *Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007, Proceedings*, ser. Lecture Notes in Computer Science, S. P. Vadhan, Ed., vol. 4392. Springer, 2007, pp. 137–156. [Online]. Available: https://doi.org/10.1007/978-3-540-70936-7_8

[83] ——, "Security against covert adversaries: Efficient protocols for realistic adversaries," *J. Cryptol.*, vol. 23, no. 2, pp. 281–343, 2010. [Online]. Available: https://doi.org/10.1007/s00145-009-9040-7

[84] M. Rivinius, P. Reisert, D. Rausch, and R. Küsters, "Publicly accountable robust multi-party computation," in *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*. IEEE, 2022, pp. 2430–2449. [Online]. Available: https://doi.org/10.1109/SP46214.2022.9833608

[85] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the Presence of Partial Synchrony," *Journal of the Association for Computing Machinery, Vol. 35, No. 2, pp.288-323*, 1988.

[86] K. P. Kihlstrom, L. E. Moser, and P. M. Melliar-Smith, "Byzantine fault detectors for solving consensus," *Comput. J.*, vol. 46, no. 1, pp. 16–35, 2003. [Online]. Available: https://doi.org/10.1093/comjnl/46.1.16

[87] A. Boldyreva, "Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme," in *Public Key Cryptography - PKC 2003, 6th International Workshop on Theory and Practice in Public Key Cryptography, Miami, FL, USA, January 6-8, 2003, Proceedings*, ser. Lecture Notes in Computer Science, Y. Desmedt, Ed., vol. 2567. Springer, 2003, pp. 31–46. [Online]. Available: https://doi.org/10.1007/3-540-36288-6_3

[88] D. Boneh, M. Drijvers, and G. Neven, "Compact multi-signatures for smaller blockchains," in *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part II*, ser. Lecture Notes in Computer Science, T. Peyrin and S. D. Galbraith, Eds., vol. 11273. Springer, 2018, pp. 435–464. [Online]. Available: https://doi.org/10.1007/978-3-030-03329-3_15

[89] D. Galindo and J. Liu, "Robust subgroup multi-signatures for consensus," in *Topics in Cryptology - CT-RSA 2022 - Cryptographers' Track at the RSA Conference 2022, Virtual Event, March 1-2, 2022, Proceedings*, ser. Lecture Notes in Computer Science, S. D. Galbraith, Ed., vol. 13161. Springer, 2022, pp. 537–561. [Online]. Available: https://doi.org/10.1007/978-3-030-95312-6_22

[90] M. Rambaud and C. Levrat, "Practical non-interactive multi-signatures, and a multi-to-aggregate signatures compiler," *IACR Cryptol. ePrint Arch.*, p. 1081, 2024. [Online]. Available: https://eprint.iacr.org/2024/1081

[91] T. Ristenpart and S. Yilek, "The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks," in *Advances in Cryptology - EUROCRYPT 2007, 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Barcelona, Spain, May 20-24, 2007, Proceedings*, ser. Lecture Notes in Computer Science, M. Naor, Ed., vol. 4515. Springer, 2007, pp. 228–245. [Online]. Available: https://doi.org/10.1007/978-3-540-72540-4_13

[92] R. Canetti, "Universally composable signature, certification, and authentication," in *17th IEEE Computer Security Foundations Workshop, (CSFW-17 2004), 28-30 June 2004, Pacific Grove, CA, USA*. IEEE Computer Society, 2004, p. 219. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/CSFW.2004.24

[93] S. Cohen, I. Keidar, and A. Spiegelman, "Not a coincidence: Sub-quadratic asynchronous byzantine agreement WHP," in *34th International Symposium on Distributed Computing, DISC 2020, October 12-16, 2020, Virtual Conference*, ser. LIPIcs, H. Attiya, Ed., vol. 179. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, pp. 25:1–25:17. [Online]. Available: https://doi.org/10.4230/LIPIcs.DISC.2020.25

[94] D. Malkhi, M. K. Reiter, A. Wool, and R. N. Wright, "Probabilistic quorum systems," *Inf. Comput.*, vol. 170, no. 2, pp. 184–206, 2001. [Online]. Available: https://doi.org/10.1006/inco.2001.3054

[95] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," *Journal of cryptology*, vol. 17, pp. 297–319, 2004.

[96] https://docs.rs/criterion/latest/criterion/.

[97] https://docs.rs/blst/latest/blst/.

[98] P. S. Barreto, B. Lynn, and M. Scott, "Constructing elliptic curves with prescribed embedding degrees," in *International conference on security in communication networks*. Springer, 2002, pp. 257–267.

[99] https://docs.rs/ed25519-dalek/latest/ed25519_dalek/.

[100] S. Das, B. Pinkas, A. Tomescu, and Z. Xiang, "Distributed randomness using weighted vufs," in *Advances in Cryptology - EUROCRYPT 2025 - 44th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Madrid, Spain, May 4-8, 2025, Proceedings, Part VII*, ser. Lecture Notes in Computer Science, S. Fehr and P. Fouque, Eds., vol. 15607. Springer, 2025, pp. 314–344. [Online]. Available: https://doi.org/10.1007/978-3-031-91098-2_12

[101] A. Haeberlen and P. Kuznetsov, "The fault detection problem," in *Principles of Distributed Systems, 13th International Conference, OPODIS 2009, Nîmes, France, December 15-18, 2009. Proceedings*, ser. Lecture Notes in Computer Science, T. F. Abdelzaher, M. Raynal, and N. Santoro, Eds., vol. 5923. Springer, 2009, pp. 99–114. [Online]. Available: https://doi.org/10.1007/978-3-642-10877-8_10

[102] H. Attiya and J. L. Welch, *Distributed computing - fundamentals, simulations, and advanced topics (2. ed.)*, ser. Wiley series on parallel and distributed computing. Wiley, 2004.

[103] B. A. Coan, "A compiler that increases the fault tolerance of asynchronous protocols," *IEEE Trans. Computers*, vol. 37, no. 12, pp. 1541–1553, 1988. [Online]. Available: https://doi.org/10.1109/12.9732

[104] A. Clement, F. Junqueira, A. Kate, and R. Rodrigues, "On the (limited) power of non-equivocation," in *Proceedings of the 2012 ACM Symposium on Principles of Distributed Computing*, ser. PODC '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 301–308. [Online]. Available: https://doi.org/10.1145/2332432.2332490

[105] C. Ho, D. Dolev, and R. van Renesse, "Making distributed applications robust," in *Principles of Distributed Systems, 11th International Conference, OPODIS 2007, Guadeloupe, French West Indies, December 17-20, 2007. Proceedings*, ser. Lecture Notes in Computer Science, E. Tovar, P. Tsigas, and H. Fouchal, Eds., vol. 4878. Springer, 2007, pp. 232–246. [Online]. Available: https://doi.org/10.1007/978-3-540-77096-1_17

[106] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*. IEEE Computer Society, 2001, pp. 136–145. [Online]. Available: https://doi.org/10.1109/SFCS.2001.959888

[107] N. A. Lynch, *Distributed Algorithms*. Elsevier, 1996.

[108] S. Coretti, J. A. Garay, M. Hirt, and V. Zikas, "Constant-round asynchronous multi-party computation based on one-way functions," in *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part II*, ser. Lecture Notes in Computer Science, J. H. Cheon and T. Takagi, Eds., vol. 10032, 2016, pp. 998–1021. [Online]. Available: https://doi.org/10.1007/978-3-662-53890-6_33

[109] J. Katz, U. Maurer, B. Tackmann, and V. Zikas, "Universally composable synchronous computation," in *Theory of Cryptography - 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings*, ser. Lecture Notes in Computer Science, A. Sahai, Ed., vol. 7785. Springer, 2013, pp. 477–498. [Online]. Available: https://doi.org/10.1007/978-3-642-36594-2_27

[110] V. Shoup and N. P. Smart, "Lightweight asynchronous verifiable secret sharing with optimal resilience," *J. Cryptol.*, vol. 37, no. 3, p. 27, 2024. [Online]. Available: https://doi.org/10.1007/s00145-024-09505-6

[111] J. Groth and V. Shoup, "Fast batched asynchronous distributed key generation," in *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part V*, ser. Lecture Notes in Computer Science, M. Joye and G. Leander, Eds., vol. 14655. Springer, 2024, pp. 370–400. [Online]. Available: https://doi.org/10.1007/978-3-031-58740-5_13

[112] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup, "Secure and efficient asynchronous broadcast protocols," in *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, ser. Lecture Notes in Computer Science, J. Kilian, Ed., vol. 2139. Springer, 2001, pp. 524–541. [Online]. Available: https://doi.org/10.1007/3-540-44647-8_31

[113] C. Cachin, K. Kursawe, and V. Shoup, "Random oracles in constantipole: practical asynchronous byzantine agreement using cryptography (extended abstract)," in *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing, July 16-19, 2000, Portland, Oregon, USA*, G. Neiger, Ed. ACM, 2000, pp. 123–132. [Online]. Available: https://doi.org/10.1145/343477.343531

[114] ——, "Random Oracles in Constantinople: Practical Asynchronous Byzantine Agreement Using Cryptography," *J. Cryptol.*, vol. 18, no. 3, pp. 219–246, 2005. [Online]. Available: https://doi.org/10.1007/s00145-005-0318-0

[115] A. Bandarupalli, X. Ji, A. Kate, C. Liu-Zhang, and Y. Song, "Concretely efficient asynchronous MPC from lightweight cryptography," *IACR Cryptol. ePrint Arch.*, p. 1666, 2024. [Online]. Available: https://eprint.iacr.org/2024/1666

[116] R. Canetti, "Universally composable security," *J. ACM*, vol. 67, no. 5, pp. 28:1–28:94, 2020. [Online]. Available: https://doi.org/10.1145/3402457

[117] M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," in *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993*, D. E. Denning, R. Pyle, R. Ganesan, R. S. Sandhu, and V. Ashby, Eds. ACM, 1993, pp. 62–73. [Online]. Available: https://doi.org/10.1145/168588.168596

[118] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," in *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, ser. Lecture Notes in Computer Science, C. Boyd, Ed., vol. 2248. Springer, 2001, pp. 514–532. [Online]. Available: https://doi.org/10.1007/3-540-45682-1_30

[119] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, "Aggregate and verifiably encrypted signatures from bilinear maps," in *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*, ser. Lecture Notes in Computer Science, E. Biham, Ed., vol. 2656. Springer, 2003, pp. 416–432. [Online]. Available: https://doi.org/10.1007/3-540-39200-9_26

[120] S. Das and L. Ren, "Adaptively secure BLS threshold signatures from DDH and co-cdh," in *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part VII*, ser. Lecture Notes in Computer Science, L. Reyzin and D. Stebila, Eds., vol. 14926. Springer, 2024, pp. 251–284. [Online]. Available: https://doi.org/10.1007/978-3-031-68394-7_9

[121] M. Andrychowicz and S. Dziembowski, "Pow-based distributed cryptography with no trusted setup," in *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, ser. Lecture Notes in Computer Science, R. Gennaro and M. Robshaw, Eds., vol. 9216. Springer, 2015, pp. 379–399. [Online]. Available: https://doi.org/10.1007/978-3-662-48000-7_19

[122] J. Katz, A. Miller, and E. Shi, "Pseudonymous secure computation from time-lock puzzles," *IACR Cryptol. ePrint Arch.*, p. 857, 2014. [Online]. Available: http://eprint.iacr.org/2014/857

Appendix A.
τ_{scr}^{++}: Scalable General Accountability

When determining how to address arbitrary Byzantine behaviors—whether to mask, detect, or punish them—it is natural to classify these behaviors [12], [101]. Essentially, faults can be categorized into two types: *omission faults*, where a party fails to send a message that should be sent, and *commission faults*, where a party sends something it should not. Commission faults can further be divided into two subcategories: *equivocation*, where a party makes contradictory statements that a correct process could not simultaneously make, and *evasion*, where a party sends messages without having received the corresponding input messages. Importantly, it has been proven that if a protocol is t-resilient, then violating safety requires at least $t+1$ commission failures from distinct Byzantine parties [12]. In other words, an attack cannot significantly exploit omission failures to evade detection after the fact.

There exists a well-studied simulation [55], [102], [103], [104], [105], traditionally attributed to Bracha [55], [102], of crash failures on top of Byzantine failures, based on reliable-broadcast [55]. In a nutshell, such a simulation can be viewed as a module θ that (1) connects the networking layer to a crash-resilient algorithm Π, and (2) only allows "benign" executions to reach Π by not forwarding any message from the networking layer to Π unless the sender's valid behavior has already been established. As a result, all Byzantine processes appear to Π as if they have crashed. More concretely, messages are sent via a secure-broadcast primitive, which guarantees that for each correct process p_i, all correct processes agree on a total order of the messages broadcast by p_i. Throughout the protocol, correct processes 'simulate' the behavior of other correct processes: they possess the necessary knowledge of the initial state, update the simulated state at each step, and to be accepted, each new message must be a legitimate output of p_i's prescribed protocol, applied to the simulated behavior up to that point.

It has been observed that Bracha's simulation, as referred to here, can be adjusted to ensure that evasion faults are *masked* (i.e., their effects are neutralized) regardless of the number of Byzantine failures, leaving *only* equivocation faults as a potential threat to safety [12]. Consequently, since reliable broadcast can tolerate up to $\lceil n/3 \rceil - 1$ Byzantine failures, it follows that any t_c-crash-resilient protocol can be automatically compiled into a protocol that tolerates up to $\min(t_c, \lceil n/3 \rceil - 1)$ Byzantine failures [102, Chapter 12], [55]. Moreover, by using an accountable version of the reliable broadcast primitive, a t_c-crash-resilient protocol can also be automatically compiled into an accountable protocol tolerating up to $\min(t_c, \lceil n/3 \rceil - 1)$ Byzantine failures [12]. The transformation described above, while general, incurs a multiplicative communication overhead, that is proportional to the communication complexity of the underlying Byzantine Reliable Broadcast primitive.

Any subquadratic Byzantine Reliable Broadcast protocol [47], [56], can be transformed using the $\mathcal{ABC}^{++}$ compiler into a subquadratic accountable version. This accountable version can then be integrated into the τ_{scr} transformation to produce τ_{scr}^{++}, an enhanced version that compiles any deterministic protocol into its accountable counterpart with subquadratic communication overhead, instead of the quadratic overhead in τ_{scr}.

Additional techniques [37], [38], [39] can further extend the applicability of such transformations when preserving hyperproperties (e.g., privacy, fairness) is required.

Appendix B.
$\mathcal{ABC}^{++}$ in the Universal Composability Framework

In this section, we provide a proof of the $\mathcal{ABC}^{++}$ transformation in the Universal Composability framework.

Universal Composability. Universal composability (UC) [106] is a widely used framework for designing, modeling, and analyzing security protocols because of its simulation-based security guarantees and its natural support for modularity. In this framework, the first step is to define an *ideal functionality* $\mathcal{F}$, which specifies the desired behavior and security properties of the protocol in an abstract manner, omitting implementation details. For example, $\mathcal{F}$ might represent a trusted and incorruptible third party that helps the parties execute the computation. Each party sends its input to the trusted party, who computes some desired function on top of as many inputs as it can expect, and returns the prescribed output to each party. The adversary's only power in this ideal world is to choose the inputs for the corrupted parties, and which inputs might be ignored due to asynchrony. In the *real world*, parties communicate by message-passing. The adversary can manipulate the order in which messages are delivered, and arbitrarily control the corrupted processes. To prove that a concrete protocol $\mathcal{P}$ *realizes* an ideal functionality $\mathcal{F}$, it must be demonstrated that for any adversary $\mathcal{A}$ attacking $\mathcal{P}$ in the *real world*, there exists a simulator $\mathcal{S}$, acting as an adversary in the *ideal world*, such that the interactions of $\mathcal{A}|\mathcal{P}$ and $\mathcal{S}|\mathcal{F}$ with an external environment $\mathcal{E}$ (serving as a distinguisher) are computationally indistinguishable. One of the core benefits of UC is its composability: once a protocol $\mathcal{P}$ is proven to realize $\mathcal{F}$, it can replace $\mathcal{P}$ as a subroutine in larger protocols without compromising security. This modular design and analysis are further guaranteed by the composition theorem inherent in the UC framework, ensuring that protocols built atop $\mathcal{P}$ remain secure even after such substitutions.

Guaranteed Output Delivery in Asynchronous UC. In the UC framework, "ensuring liveness" is nuanced due to polynomially bounded executions. Traditionally, a property is said to *eventually* hold if, for every infinite execution where each machine/task is activated infinitely often, the property is not indefinitely false [107, Chapter 8.5].

In the framework of [108], adapted from [109], a party must actively request an output from the functionality $\mathcal{F}$. The simulator $\mathcal{S}$ can delay the output by ignoring requests for a polynomial number of activations. If the environment activates the party sufficiently often, the party will eventually receive its output from $\mathcal{F}$. This guarantees termination, ensuring all honest parties eventually obtain their outputs if the environment allocates enough resources.

Instead, we adopt the framework proposed by Shoup et al. [71], [110], [111], [112], [113], [114], which reinterprets the notion of "eventually holds" by requiring that the property holds if and when all messages sent from one honest party to another are eventually delivered. However, without a constraint on the total message complexity, liveness properties could be trivially satisfied by protocols that perpetually generate new messages without delivering earlier ones. This motivates the need for *efficient* protocols, where communication and computational complexities are uniformly bounded. These bounds are fixed polynomials in the security parameter and the number of parties, holding with all but negligible probability for any polynomially bounded adversary. This efficiency is preserved under protocol composition and simplifies the definition of *guaranteed output delivery*. Clearly, we only propose efficient protocols.

Following [20], [115], we use the term *delayed output* to describe an output whose delivery is triggered by the adversary after an arbitrary amount of time.

Accountable Universal Composability (AUC). The AUC framework [18] extends UC with accountability requirements. Functionalities additionally define how security violations (e.g., agreement breaches) must expose malicious parties. In our case, each accountable functionality is associated with a *fair public judge*—a protocol, locally executable by any process, producing *verdicts* blaming parties using *publicly verifiable evidence*. We consider judges ensuring (1) *fairness*: honest parties are never falsely blamed, (2) *external verifiability*: third parties can validate evidence, and (3) *individual accountability*: verdicts take the form $\mathsf{dis}(D)$, meaning that all processes in the set D are deemed dishonest. Each correct process runs a local instance of such a (fair and public) judge.

Proof structure. We begin by proving that $\mathcal{P}_{\mathrm{rat}}$ (Algorithm 8) realizes the ratifier's ideal functionality $\mathcal{F}_{\mathrm{rat}}$ defined in §B.1. Next, we rely on a result from [15], stated in the 2020 revision of Canetti's UC framework [116], which establishes that the propagator's ideal functionality $\mathcal{F}_{\mathrm{prop}}$ can be directly realized. We then show that the sequential composition $\mathcal{F}_{\mathrm{rat}} \rhd \mathcal{F}_{\mathrm{prop}}$ realizes the accountable confirmer functionality $\mathcal{F}_{\mathrm{conf}}^{\mathrm{acc}}$. Finally, we describe the $\mathcal{ABC}^{++}$ compiler, which sequentially composes any easily accountable functionality $\mathcal{F}_{\mathrm{easy}}$ (cf. Definition 3)—such as consensus, reliable broadcast, consistent broadcast, or one-to-many ZKPs—with $\mathcal{F}_{\mathrm{conf}}^{\mathrm{acc}}$ to obtain its accountable counterpart. The realizations rely on ideal functionalities $\mathcal{F}_{\mathrm{vcs}}$ and $\mathcal{F}_{\mathrm{msig}}$ for verifiable committee sampling and multi-signatures, presented in §C and §D, respectively.

B.1. The Ratifier Functionality $\mathcal{F}_{\mathrm{rat}}$

B.1.1. $\mathcal{F}_{\mathrm{rat}}$'s Specification. When an easily accountable functionality (e.g., $\mathcal{F}_{\mathrm{vba}}$, $\mathcal{F}_{\mathrm{rbc}}$, or $\mathcal{F}_{\mathrm{zkPoK}}^{1:\mathrm{M}}$) yields a (pre-)decision v, processes must determine whether they unanimously agree on v. Furthermore, this confirmation must be accompanied by an externally verifiable *certificate*, enabling an external observer (such as a judge instance) to detect inconsistencies and identify dishonest processes when conflicting certificates are presented. These requirements are formally captured by the ideal functionality $\mathcal{F}_{\mathrm{rat}}$, introduced in Functionality 1.

Namely, $\mathcal{F}_{\mathrm{rat}}$ ensures the following properties:

- **Integrity**: For any $f \in [0:n]$, a correct process cannot confirm a value that it did not submit.
- **Optimistic convergence**: If $f \leq t$ and all correct process submit the same value v, then every correct process eventually confirms that value.
- **Validity**: If a so-far correct process triggers $(\mathsf{confirm}, \mathsf{sid}, \mathsf{ssid}, m, c)$, then $\mathsf{valid}(c) = \mathit{true}$.
- **Forensic support for agreement**: If two correct processes p_i and p_j output different values, they respectively store conflicting certificates c_i and c_j, such that $\mathsf{Judge}(c_i, c_j)$ returns the verdict $\mathsf{dis}(D)$, exposing $|D| \geq d$ dishonest parties.

B.1.2. $\mathcal{F}_{\mathrm{rat}}$'s Realization. We present the $\mathcal{F}_{\mathrm{rat}}$'s realization $\mathcal{P}_{\mathrm{rat}}$ (see Algorithm 8). It refers to the (public and fair) judge described in Algorithm 6 and the $\mathcal{F}_{\mathrm{valid\_full}}$ functionality described in Algorithm 7. Two certificates are conflicting if they refers two conflicting submit messages with different (hash) values.

To implement $\mathcal{F}_{\mathrm{rat}}$, a VRF-selected committee of expected size λ broadcasts a SUBMIT message containing a hash of their signed pre-decided value to all other processes. Once a process receives at least $W_\delta^{\epsilon,\lambda} \simeq 2\lambda/3$ valid SUBMIT messages supporting the (hashed) pre-decision v, it confirms v and stores the signed messages from a "VRF-quorum" of size $W_\delta^{\epsilon,\lambda}$ for future verification. Optimistic convergence is ensured by the first Chernoff bound in Theorem 1.

In case of an agreement violation, two processes will store conflicting *certificates*, each consisting of $W_\delta^{\epsilon,\lambda}$ signed SUBMIT messages from a VRF-selected committee but for differing (hashed) values. These two certificates serve as undeniable

Functionality 1 $\mathcal{F}_{\text{rat}}$

Parameters:

t: Optimistic Threshold
t_{acc}: Pessimistic Threshold
d: Number of dishonest processes that should be exposed in case of disagreement
$\mathsf{valid}(\cdot)$: Predicate over certificates
$\mathsf{conflict}(\cdot,\cdot)$: Predicate over pairs of certificates ▷ Later, composition will require propagation-friendliness (Definition 1) of $(\mathsf{valid}, \mathsf{conflict})$

Variables:

$\mathsf{Set}(\mathsf{Message})$ *submitted* ▷ Set of submitted messages
$\mathsf{Map}(\mathsf{Message} \to \mathsf{Set}(\mathsf{Process}))$ *supporters* ▷ Maps each message m to the set of processes submitting m.

Participants:

$p_1, \ldots, p_n$: Parties interacting with the functionality.
Sim: Ideal adversary (simulator).
Judge: (public and fair) judge

Events:

request $(\mathsf{submit}, \mathsf{sid}, \mathsf{ssid}, m)$: A party submits a message m.
notification $(\mathsf{confirm}, \mathsf{sid}, \mathsf{ssid}, m, c)$: A party confirms the message m, backed by a certificate c.

Functionality:

Upon receiving $(\mathsf{submit}, \mathsf{sid}, \mathsf{ssid}, m)$ from an uncorrupted process p_i:
  $submitted \leftarrow submitted \cup \{m\}$
  $supporters[m] \leftarrow supporters[m] \cup \{p_i\}$
  Notify Sim of the submission of m by p_i

If $\exists m$ s.t. $submitted = \{m\} \wedge |supporters[m] \cap \mathsf{Correct}| \geq n - t$:
  Send $(\mathsf{confirm}, \mathsf{sid}, \mathsf{ssid}, m, c_i)$ to every process p_i as a delayed output, such that $\mathsf{valid}(c_i) = true$ ▷ Agreement and Liveness

If $\exists m, m'$ s.t. $m, m' \in submitted$, with $m \neq m'$: ▷ Liveness and Agreement are not guaranteed anymore
  Forward any $(\mathsf{confirm}, \mathsf{sid}, \mathsf{ssid}, m'', c_i)$ from Sim to any targeted process $p_i \in supporters[m'']$, such that $\mathsf{valid}(c_i) = true$

// Degraded Mode ($f \leq t_{acc}$)
If two correct processes deliver $(\mathsf{confirm}, \mathsf{sid}, \mathsf{ssid}, m, c)$ and $(\mathsf{confirm}, \mathsf{sid}, \mathsf{ssid}, m' \neq m, c')$, respectively, then $\mathsf{conflict}(c, c') = true$.
// If $(\mathsf{valid}, \mathsf{conflict})$ is propagation-friendly w.r.t. (Judge, d), the judge would return the (fair) verdict $\mathsf{dis}(D)$ on input (c, c'), where $|D| \geq d$

Algorithm 6 Public and fair judge J_{rat} on top of $\mathcal{F}_{\text{msig}}|\mathcal{F}_{\text{CA}}$

Uses:
  $\mathcal{F}_{\text{msig}}$ ▷ Multi-signature scheme (see Functionality 5 in §D.2)
  $\mathcal{F}_{\text{CA}}$ ▷ Certification Authority (see [92, §3.2])

upon receiving $(\mathsf{proof}, \mathsf{sid}, \mathsf{ssid}, \pi)$
  parse π as $((h_1, G_1, \sigma_1), (h_2, G_2, \sigma_2))$
  Request $(\mathsf{retrieve}, \mathsf{sid}, G_1 \cup G_2)$ from $\mathcal{F}_{\text{CA}}$

upon receiving $(\mathsf{retrieve}, \mathsf{sid}, \{(p_k, pk_k^{msig}, *)\}_{p_k \in G_1 \cup G_2}$ from $\mathcal{F}_{\text{CA}}$:
  Request $(\mathsf{verify}, \mathsf{sid}, \mathsf{ssid}, G_1, \langle \text{SUBMIT}, h_1 \rangle, \sigma_1, \{pk_i^{msig}\}_{i \in G_1})$ from $\mathcal{F}_{\text{msig}}$
  Request $(\mathsf{verify}, \mathsf{ssid}, \mathsf{sid}, G_2, \langle \text{SUBMIT}, h_2 \rangle, \sigma_2, \{pk_j^{msig}\}_{j \in G_2})$ from $\mathcal{F}_{\text{msig}}$

upon receiving from $\mathcal{F}_{\text{msig}}$: $(\mathsf{verified}, \mathsf{sid}, \mathsf{ssid}, G_1, \langle \text{SUBMIT}, h_1 \rangle, 1)$ and $(\mathsf{verified}, \mathsf{sid}, \mathsf{ssid}, G_2, \langle \text{SUBMIT}, h_2 \rangle, 1)$ with $h_1 \neq h_2$:
  return $\mathsf{dis}(G_1 \cap G_2)$. ▷ No correct process ever signs two conflicting SUBMIT messages

proof of misbehavior against processes that signed conflicting SUBMIT messages. With overwhelming probability, at least $B^{\epsilon,\lambda}_{\delta,\hat{\delta}} \simeq \lambda/3$ processes will be provably dishonest, as supported by the second Chernoff bound in Theorem 1.

Lemma 1. $\mathcal{P}_{\mathsf{rat}}$ (Algorithm 8), realizes the ideal functionality $\mathcal{F}_{\text{rat}}$ (see Functionality 1) in the ($\mathcal{F}_{\text{vcs}}$, $\mathcal{F}_{\text{msig}}$, $\mathcal{F}_{\text{CA}}$, $\mathcal{F}_{\text{auth}}$)-hybrid model, with t-resiliency, 1 round, and an expected message complexity of $O(\lambda n)$, each message being of size $O(\kappa)$. Here, both $\mathcal{P}_{\mathsf{rat}}$ and $\mathcal{F}_{\text{rat}}$ refer to the judge J_{rat} (see Algorithm 6) and the valid_full predicate (see $\mathcal{F}_{\text{valid\_full}}$ functionality in Algorithm 7).

PROOF.

Foreword. We consider the ideal functionality

$$\mathcal{F}_{\text{rat}} = p_1 \mid \cdots \mid p_n \mid \tilde{\mathcal{F}}_{\text{rat}}$$

where processes $p_1, \ldots, p_n$ represent the dummy processes, and its realization in the $(\mathcal{F}_{\text{vcs}}, \mathcal{F}_{\text{msig}}, \mathcal{F}_{\text{CA}}, \mathcal{F}_{\text{auth}})$-hybrid model

$$\mathcal{P}_{\text{rat}} = \bar{p}_1 \mid \cdots \mid \bar{p}_n \mid \mathcal{F}_{\text{vcs}} \mid \mathcal{F}_{\text{msig}} \mid \mathcal{F}_{\text{CA}} \mid \mathcal{F}_{\text{auth}}.$$

Algorithm 7 $\mathcal{F}_{\text{valid\_full}}$ on top of $\mathcal{F}_{\text{msig}}|\mathcal{F}_{\text{CA}}|\mathcal{F}_{\text{vcs}}$

Uses:
  $\mathcal{F}_{\text{msig}}$ ▷ Multi-signature scheme (see Functionality 5 in §D.2)
  $\mathcal{F}_{\text{CA}}$ ▷ Certification Authority (see [92, §3.2])
  $\mathcal{F}_{\text{vcs}}$ ▷ Verifiable Committee Sampling (see Functionality 4 in §C.2)

upon receiving $(\mathsf{verify}, \mathsf{sid}, \mathsf{ssid}, c)$
  parse c as $\big(h, G, \tilde{\sigma}, (\sigma_1, \ldots, \sigma_{|G|})\big)$
  Request $(\mathsf{retrieve}, \mathsf{sid}, G)$ from $\mathcal{F}_{\text{CA}}$

upon receiving $(\mathsf{retrieve}, \mathsf{sid}, \{(p_i, pk_i^{msig}, pk_i^{vcs})\}_{p_i \in G}$ from $\mathcal{F}_{\text{CA}}$:
  Request $(\mathsf{verify}, \mathsf{sid}, \mathsf{ssid}, G, \langle \text{SUBMIT}, h\rangle, \tilde{\sigma}, \{pk_i^{msig}\}_{p_i \in G})$ from $\mathcal{F}_{\text{msig}}$
  Request $(\mathsf{verify}, \mathsf{sid}, \mathsf{ssid}, G, \{\sigma_i, pk_i^{vcs}\}_{p_i \in G})$ from $\mathcal{F}_{\text{vcs}}$

upon receiving $(\mathsf{verified}, \mathsf{sid}, \mathsf{ssid}, G, \langle \text{SUBMIT}, h\rangle, 1)$ from $\mathcal{F}_{\text{msig}}$ and $(\mathsf{verification}, \mathsf{sid}, \mathsf{ssid}, G, 1)$ from $\mathcal{F}_{\text{vcs}}$:
  return $(\mathsf{verified}, \mathsf{sid}, \mathsf{ssid}, c, 1)$

Algorithm 8 $\mathcal{P}_{\text{rat}} \leq \mathcal{F}_{\text{rat}}$ on top of $\mathcal{F}_{\text{msig}}|\mathcal{F}_{\text{CA}}|\mathcal{F}_{\text{vcs}}$: Pseudocode (for process p_i)

Uses:
  $\mathcal{F}_{\text{vcs}}$ parametrized with probability λ/n ▷ Verifiable committee sampling (see Functionality 4 in §C.2)
  $\mathcal{F}_{\text{CA}}$ ▷ Certification Authority $\mathcal{F}_{\text{CA}}$ [92, §3.2]
  $\mathcal{F}_{\text{msig}}$ ▷ Multi-signature scheme (see Functionality 5 in §D.2). Public key publication is handled by $\mathcal{F}_{\text{CA}}$.
  $\mathcal{F}_{\text{auth}}$ ▷ Authenticated network functionality, UC-realizable in the $(\mathcal{F}_{\text{CA}}, \mathcal{F}_{(\text{m})\text{sig}})$-hybrid model [92, §4].

Local Variables:
  $\mathsf{Dictionary}(\mathsf{Process} \times \mathsf{Hash\_Value} \to \mathsf{Signature})$ $from\_unverified_i \leftarrow \perp$
  $\mathsf{Dictionary}(\mathsf{Process} \times \mathsf{Hash\_Value} \to \mathsf{Signature})$ $from_i \leftarrow \perp$
  $\mathsf{Dictionary}(\mathsf{Process} \times \mathsf{Hash\_Value} \to \mathsf{VRF\_Proof})$ $vrf\_proofs_i \leftarrow \perp$
  $\mathsf{Message}$ $predecided_i \leftarrow \perp$
  $\mathsf{Signature}$ $\sigma_i \leftarrow \perp$
  $\mathsf{VRF\_Proof}$ $\pi_i \leftarrow \perp$

upon $(\mathsf{submit}, \mathsf{sid}, \mathsf{ssid}, v)$:
  $predecided_i \leftarrow v$
  Request $(\mathsf{eligibility}, \mathsf{sid}, \mathsf{ssid})$ from $\mathcal{F}_{\text{vcs}}$

upon receiving $(\mathsf{eligibility}, \mathsf{sid}, \mathsf{ssid}, coin_i, \pi_i')$ from $\mathcal{F}_{\text{vcs}}$:
  if $coin_i = 1$:
    $\pi_i \leftarrow \pi_i'$
    Request $(\mathsf{sign}, \mathsf{sid}, \mathsf{ssid}, p_i, \langle \text{SUBMIT}, \mathsf{hash}(v)\rangle)$ from $\mathcal{F}_{\text{msig}}$

upon receiving $(\mathsf{signature}, \mathsf{sid}, \mathsf{ssid}, p_i, \langle \text{SUBMIT}, \mathsf{hash}(v)\rangle, \sigma_i)$ from $\mathcal{F}_{\text{msig}}$:
  send $\big(\mathsf{broadcast}, \mathsf{sid}, \mathsf{ssid}, \langle \text{SUBMIT}, \mathsf{hash}(v)\rangle, \sigma_i, \pi_i\big)$ to $\mathcal{F}_{\text{auth}}$

upon receiving $\big(\mathsf{rcv}, \mathsf{sid}, \mathsf{ssid}, p_j, \langle \text{SUBMIT}, h\rangle, \sigma_j, \pi_j\big)$ from $\mathcal{F}_{\text{auth}}$:
  $from\_unverified_i[p_j, h] \leftarrow \sigma_j$
  $vrf\_proofs_i[p_j, h] \leftarrow \pi_j$
  Request $(\mathsf{verify}, \mathsf{sid}, \mathsf{ssid}, \{p_j\}, \langle \text{SUBMIT}, h\rangle, \sigma_j, \{pk_j\})$ from $\mathcal{F}_{\text{msig}}$ ▷ pk_j is obtained from $\mathcal{F}_{\text{CA}}$
  Request $(\mathsf{verify}, \mathsf{sid}, \mathsf{ssid}, p_j, \pi_j)$ from $\mathcal{F}_{\text{vcs}}$

upon receiving $(\mathsf{verified}, \mathsf{sid}, \mathsf{ssid}, \{p_j\}, \langle \text{SUBMIT}, h\rangle, 1)$ from $\mathcal{F}_{\text{msig}} \wedge (\mathsf{verified}, \mathsf{sid}, \mathsf{ssid}, p_j, 1)$ from $\mathcal{F}_{\text{vcs}}$:
  $from_i[p_j, h] \leftarrow from\_unverified_i[p_j, h]$

upon $|G := \{p_j \mid from_i[p_j, \mathsf{hash}(predecided_i)] \neq \perp\}| \geq W_\delta^{\epsilon,\lambda}$:
  Request $(\mathsf{cb}, \mathsf{sid}, \mathsf{ssid}, G, \mathsf{hash}(predecided_i), \{(\sigma_j, pk_j)\}_{p_j \in G})$ from $\mathcal{F}_{\text{msig}}$ ▷ $\{pk_j\}_{p_j \in G}$ is obtained from $\mathcal{F}_{\text{CA}}$

upon receiving $(\mathsf{multisignature}, \mathsf{sid}, \mathsf{ssid}, G, \mathsf{hash}(predecided_i), \sigma_G)$ from $\mathcal{F}_{\text{msig}}$:
  $light\_cert_i \leftarrow (\mathsf{hash}(predecided_i), G, \sigma_G)$
  $\pi_i^{poe} \leftarrow \{vrf\_proofs[p_j, \mathsf{hash}(predecided_i)] | p_j \in G\}$
  $full\_cert_i \leftarrow (light\_cert_i, \pi_i^{poe})$
  trigger $(\mathsf{confirm}, \mathsf{sid}, \mathsf{ssid}, predecided_i, full\_cert_i)$

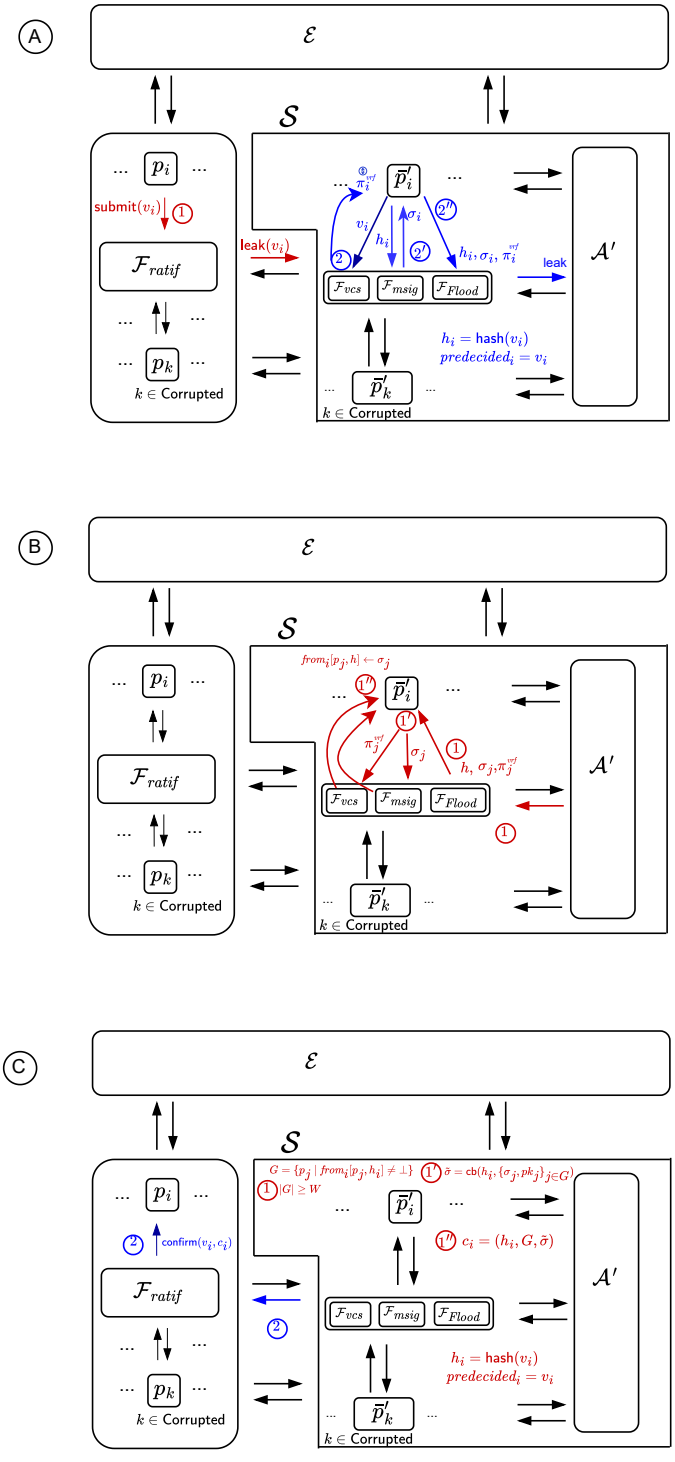

Figure 1: $\mathcal{P}_{\text{rat}}$ realizes $\mathcal{F}_{\text{rat}}$

Let $\mathcal{A}$ be the adversary of $\mathcal{P}_{\text{rat}}$. The simulator $\mathcal{S}$ internally simulates the adversary, denoted by $\mathcal{A}'$. Additionally, for *each* $k \in [1:n]$, it simulates $\bar{p}_k$, denoted as $\bar{p}'_k$. Let us note it can do so, since no information is private in the protocol. Finally, the simulator simulates the functionalities $\mathcal{F}_{\text{vcs}}$, $\mathcal{F}_{\text{msig}}$, $\mathcal{F}_{\text{CA}}$ and $\mathcal{F}_{\text{auth}}$. This includes selecting its own randomness for the simulated functionality $\mathcal{F}_{\text{vcs}}$. Interaction with the certification authority $\mathcal{F}_{\text{CA}}$ [92, §3.2], binding each process ID with a unique public key, is omitted for sake of simplicity.

Messages received by $\mathcal{S}$ from the environment $\mathcal{E}$ are forwarded to $\mathcal{A}'$. Likewise, any messages sent by $\mathcal{A}'$ that are destined for the environment are forwarded to $\mathcal{E}$. The simulator $\mathcal{S}$ keeps the corruption status synchronized; that is, whenever $\mathcal{A}$ corrupts a process $\bar{p}'_k$, the simulator also corrupts the corresponding process p_k. Incoming messages from a corrupted process p_k to $\mathcal{S}$ are forwarded to $\mathcal{A}'$ on behalf of the corresponding corrupted process $\bar{p}'_k$. Conversely, whenever a corrupted process $\bar{p}'_k$ attempts to output a message to the environment, $\mathcal{S}$ instructs the corresponding corrupted process p_k to output the same message to $\mathcal{E}$.

We now describe the strategy of the simulator, beyond the foreword. The process consists of three main situations, labeled A, B, and C, depicted in Figure 1.

A: Honest submission. When a correct process p_i sends a submission request $(\mathsf{submit}, \mathsf{sid}, \mathsf{ssid}, v_i)$ to $\mathcal{F}_{\text{rat}}$, the simulator $\mathcal{S}$ is notified through the leak interface. It immediately simulates the corresponding actions of $\bar{p}'_i$:

- Request eligibility from $\mathcal{F}_{\text{vcs}}$ on behalf of p_i, and observe both the eligibility coin and the eligibility proof π_i^{vrf}.
- If the coin indicates eligibility, request an individual signature from $\mathcal{F}_{\text{msig}}$ on $\langle \text{SUBMIT}, h_i \rangle$ with $h_i = \mathsf{hash}(v_i)$.
- Once the signature σ_i is obtained from the leak tape, broadcast the message $\langle \mathsf{submit}, h_i, \sigma_i, \pi_i^{vrf} \rangle$ through $\mathcal{F}_{\text{auth}}$ on behalf of process $\bar{p}'_i$.

B: Simulating receipt of adversarial messages. Whenever $\mathcal{A}'$ instructs the simulated $\mathcal{F}_{\text{auth}}$ to deliver a message

$$(\mathsf{rcv}, \mathsf{sid}, \mathsf{ssid}, p_j, \langle \text{SUBMIT}, h \rangle, \sigma_j, \pi_j)$$

to a simulated honest party $\bar{p}'_i$, the simulator verifies:

- Whether σ_j is a valid signature for $\langle \text{SUBMIT}, h \rangle$ under p_j's key using $\mathcal{F}_{\text{msig}}$,
- Whether π_j is a valid proof of p_j's eligibility using $\mathcal{F}_{\text{vcs}}$.

If both verifications succeed, $\mathcal{S}$ updates $\mathit{from}_i[p_j, h]$ with σ_j, simulating the update at line 28 of the protocol. Additionally, the simulator stores the associated eligibility proof π_j in $\mathit{vrf\_proofs}_i[p_j, h]$.

C: Confirmation triggered by a simulated process. Suppose the simulated process $\bar{p}'_i$ collects enough valid and eligible signatures for a value v_i such that the threshold in line 29 is met. It proceeds to:

- Request a combined signature σ_G from $\mathcal{F}_{\text{msig}}$,
- Form the full certificate $\tilde{c}_i = (c_i, \pi_i^{poe})$, with $c_i = (\mathsf{hash}(v_i), G, \sigma_G)$, and $\pi_i^{poe} = \{\mathit{vrf\_proofs}_i[p_j, \mathsf{hash}(v_i)] | p_j \in G\}$
- Trigger the confirm output $(\mathsf{confirm}, \mathsf{sid}, \mathsf{ssid}, v_i, \tilde{c}_i)$ internally.

At this point, the simulator $\mathcal{S}$ instructs the ideal functionality $\mathcal{F}_{\text{rat}}$ to forward the same delayed output to p_i.

Forensic support in the degraded case. Suppose two correct processes p_i and p_j confirm conflicting values $v_i \neq v_j$ based on internally simulated executions. By construction, both must hold valid certificates c_i and c_j with multisignatures from respective signer sets Q_i and Q_j, each of size at least $W = W_\delta^{\epsilon,\lambda}$. Thus, the J_{rat} would return $G = Q_i \cap Q_j$. Since signatures are only stored after successful verification of both eligibility and signature validity, each signer must be in the eligible set of the corresponding sub-session. Due to Theorem 1 (forensic), with overwhelming probability $(1 - \mathit{neg}(\lambda))$, G has size at least $B_{\delta,\hat{\delta}}^{\epsilon,\lambda}$.

Liveness in the nominal case. Assume at most t parties are corrupted, and all correct parties submit the same value v. Then, by Theorem 1 (liveness), with overwhelming probability $(1 - \mathit{neg}(\lambda))$, at least $W = W_\delta^{\epsilon,\lambda}$ correct parties will be eligible and will successfully broadcast a submit message with valid signature and eligibility proof. These will be delivered to all correct processes, and stored in their local dictionaries from_i and $\mathit{vrf\_proofs}_i$, allowing them to construct valid certificates and trigger confirmation of v.

Conclusion. In both the nominal and degraded cases, the behavior of the simulated execution of $\mathcal{P}_{\text{rat}}$ in the $(\mathcal{F}_{\text{vcs}}, \mathcal{F}_{\text{msig}}, \mathcal{F}_{\text{CA}}, \mathcal{F}_{\text{auth}})$-hybrid world is indistinguishable from an ideal execution of $\mathcal{F}_{\text{rat}}$. Hence:

$$\mathcal{P}_{\text{rat}} \text{ realizes } \mathcal{F}_{\text{rat}} \quad \text{in the } (\mathcal{F}_{\text{vcs}}, \mathcal{F}_{\text{msig}}, \mathcal{F}_{\text{CA}}, \mathcal{F}_{\text{auth}})\text{-hybrid model.}$$

□

Functionality 2 $\mathcal{F}_{\text{prop}}$

Parameters:
t_{acc}: Pessimistic Threshold
d: Number of dishonest processes that should be exposed in case of disagreement
$\mathsf{valid}(\cdot)$: Predicate over strings
$\mathsf{conflict}(\cdot,\cdot)$: Predicate over pairs of strings

Variables:
$\mathsf{Set}(\mathsf{Certificate})$ *Prop* ▷ Set of propagated certificates

Participants:
$p_1, \ldots, p_n$: Parties interacting with the functionality.
Sim: Ideal adversary (simulator).
Judge: (public and fair) judge

Events:
request $(\mathsf{propagate}, \mathsf{sid}, \mathsf{ssid}, cert \in \{c \in \mathsf{String} | \mathsf{valid}(c)\})$: A party propagates a valid certificate
notification $(\mathsf{detection}, \mathsf{sid}, \mathsf{ssid}, D, \pi)$: A party outputs a proof π, exposing a set D of dishonest processes

Functionality:
Upon receiving $(\mathsf{propagate}, \mathsf{sid}, \mathsf{ssid}, cert)$:
$Prop \leftarrow Prop \cup \{cert\}$
Notify Sim accordingly

If $f \leq t_{acc} \wedge \exists c, c' \in Prop$, such that $\mathsf{conflict}(c, c') = true$
Send $(\mathsf{detection}, \mathsf{sid}, \mathsf{ssid}, D_i, \pi_i)$ to each p_i as a delayed output, s.t. $\mathsf{Judge}(\pi_i)$ returns $\mathsf{Dis}(D_i)$ with $|D_i| \geq d$

B.2. The Propagator Functionality $\mathcal{F}_{\text{prop}}$

As discussed earlier, forensic support does not specify how the judge acquires the necessary pieces of evidence to detect misbehavior. This task is handled by the propagator functionality $\mathcal{F}_{\text{prop}}$, defined in Functionality 2.

This functionality ensures:

- **Accountability for conflict-freeness**: If two correct processes collectively propagate a pair of certificates (c, c') verifying $\mathsf{conflict}(c, c') = true$, then, every correct process p_i eventually gets a proof π_i exposing d dishonest processes.

A naive implementation would require every correct process to broadcast its certificate, resulting in quadratic communication overhead. Instead, we propose $\Pi^{prop}_{\mathsf{ERFlood}}$ (Algorithm 9), a very minor modification of the flooding protocol Π_{ERFlood} described in [15]. The protocol's pseudocode is simple (though its analysis is far from trivial): upon receiving a verifiably-valid message m, each party flips a coin for each other process that decides, with some probability ρ^*, if m should be relayed to this process. When ρ^* is sufficiently large, Π_{ERFlood} guarantees the formation of a connected Erdős–Rényi graph with a low diameter.

In $\Pi^{prop}_{\mathsf{ERFlood}}$, which is based on Π_{ERFlood}, a message m containing a valid certificate $(h, G, \sigma) \in \mathsf{Certificate}$ is relayed by process p_i only if it has not already relayed m *or* another message m' with a certificate $(h', G', \sigma') \in \mathsf{Certificate}$ where $h' = h$. The modification (excluding the accountability layer) is in red, line 19. It is easy to see that the analysis of $\Pi^{prop}_{\mathsf{ERFlood}}$ mirrors the (non-trivial) analysis of Π_{ERFlood}.

Lemma 2. Let $x \in \{1, 2\}$. Let $\mathsf{para} = (n, \kappa, t, t_{acc}, \gamma, \epsilon, \delta, \hat{\delta}, \rho_x)$ be a x-scalable parametrization. $\Pi^{prop}_{\mathsf{ERFlood}}\langle\rho_x\rangle$ (Algorithm 9) realizes $\mathcal{F}_{\text{prop}}$ (see Functionality 2) against a 1-delayed-adaptive adversary, with expected round complexity rc_x, and expected communication complexity mc_x, where each message is of size $\mathsf{ms} = O(\lambda\kappa)$, such that $(\mathsf{mc}_x, \mathsf{rc}_x, d)$ is a x-scalable tuple w.r.t. para.

PROOF. Without loss of generality, we consider the set $\mathfrak{A}_{prop}$ of adversaries for $\mathcal{P}_{\text{prop}}$ that never corrupt p_1 or p_2, and the set $\mathfrak{E}_{prop}$ of environments $\mathcal{E}_{prop}$ for machines of the form $\mathcal{P}_{\text{prop}} \mid \mathcal{A}$ with $\mathcal{A} \in \mathfrak{A}_{prop}$. We further restrict $\mathfrak{E}_{prop}$ to environments that only order processes to propagate valid certificates: p_1 propagates c_1, p_2 propagates c_2, where $\mathsf{conflict}(c_1, c_2) = true$. We fix the session identifiers sid and ssid. Finally, environments in $\mathfrak{E}_{prop}$ must halt and return a decision in $\{0, 1\}$ if it receives a detection notification from a so-far correct process.

We note $c \equiv c'$ iff $\mathsf{valid}(c) = \mathsf{valid}(c') = true$ $\mathsf{conflict}(c, c') = false$. We define C_1, C_2 as the sets of valid certificates equivalent (i.e., non-conflicting) with c_1 and c_2, respectively. Let C_3 be the set of valid certificates that are not in $C_1 \cup C_2$—that is, certificates conflicting with both c_1 and c_2. Let c_3 denote the first element of C_3 according to a fixed lexicographic order.

Let $\mathcal{A}_{prop} \in \mathfrak{A}_{prop}$. We define an adversary $\mathcal{A}_{flood}$ for Π_{ERFlood} that behaves exactly as $\mathcal{A}_{prop}$, with one exception: when $\mathcal{A}_{flood}$ is notified via $\mathcal{F}_{\text{MT}}$ that process p_i has requested to send certificate c_i to process p_j, but $\mathcal{A}_{flood}$ has already ordered $\mathcal{F}_{\text{MT}}$ to deliver to p_j a certificate $c' \equiv c_i$, then $\mathcal{A}_{flood}$ stores (p_j, c_i) in a set Delayed. For any pair $(p, c) \in \mathsf{Delayed}$, $\mathcal{A}_{flood}$ delays p's reception of c as long as any messages remain in flight. We note $\mathfrak{E}_{flood}$ the set of environments for $\Pi_{\mathsf{ERFlood}} | \mathcal{A}_{flood}$

Algorithm 9 $\Pi^{prop}_{\mathsf{ERFlood}}\langle\rho^*\rangle \leq \mathcal{F}_{\text{prop}}$: Pseudocode (for process p_i)

Uses:
  $\mathcal{F}_{\text{MT}}$ ▷ Message Transfer Functionality
  $\mathcal{F}_{\text{valid\_full}}$ ▷ See Algorithm 7
  J_{rat} ▷ Judge (see Algorithm 6)

Parameters:
  Integer $\delta, \hat{\delta}, \epsilon, \lambda \in \Omega(1)$
  Integer $d = B^{\epsilon,\lambda}_{\delta,\hat{\delta}}$: Number of dishonest processes that should be exposed in case of disagreement

Local Variables:
  Set(Certificate) $certificates_i \leftarrow \emptyset$

upon relay(sid, ssid, $cert$):
  $\mathcal{N}_{new} \xleftarrow{\$} \mathcal{N}(\rho^*)$, where $\mathcal{N}(\rho^*)$ is the random variable that returns a subset of processes in Ψ, where the inclusion of a process follows the Bernouilli distribution with expected value ρ^*.
  $certificates_i \leftarrow certificates_i \cup \{cert\}$
  Request (multicast, sid, ssid, $\mathcal{N}_{new}$, $\langle$RELAY, $cert\rangle$) from $\mathcal{F}_{\text{MT}}$ ▷ p_i sends $n\rho^*$ messages on expectation

upon (propagate, sid, ssid, $cert$):
  trigger relay(sid, ssid, $cert$)

upon receiving (deliver, sid, ssid, p', $\langle$RELAY, $cert'\rangle$) from $\mathcal{F}_{\text{MT}}$:
  Request (verify, sid, ssid, $cert'$) from $\mathcal{F}_{\text{valid\_full}}$: ▷ invalid certificates are ignored
    when receiving (verified, sid, ssid, $cert'$, 1) from $\mathcal{F}_{\text{valid\_full}}$:
      if $certificates_i = \emptyset$: ▷ There is no need to relay several non-conflicting certificates
        trigger relay(sid, ssid, $cert'$)
  //Accountability layer (code below has been added to Π_{ERFlood})
      if $\exists cert \in certificates_i$, such that $J_{\mathsf{rat}}(cert, cert')$ returns Dis(D) with $|D| \geq d$
        Request (broadcast, sid, ssid, $\langle$PROOF, $(cert, cert')\rangle$) from $\mathcal{F}_{\text{MT}}$

upon receiving (deliver, sid, ssid, p_j, $\langle$PROOF, $\pi\rangle$) from $\mathcal{F}_{\text{MT}}$:
  if $J_{\mathsf{rat}}(\pi)$ returns Dis(D) with $|D| \geq d$:
    trigger (detection, sid, ssid, D, π)

Since [15, Theorem 3] states that Π_{ERFlood} UC-emulates $\mathcal{F}_{\text{flood}}$, there exists a simulator $\mathcal{S}_{flood}$ such that,

$$\forall \mathcal{E} \in \mathfrak{E}_{flood},\ \mathcal{E}|\Pi_{\mathsf{ERFlood}} \mid \mathcal{A}_{flood} \simeq \mathcal{E}|\mathcal{F}_{\text{flood}} \mid \mathcal{S}_{flood}.$$

We now define a simulator $\mathcal{S}_{prop}$ for $\mathcal{F}_{\text{prop}}$ that behaves exactly as $\mathcal{S}_{flood}$, with the following exception (besides the interpretation of propagation requests as flooding requests): when $\mathcal{S}_{flood}$ triggers the delivery of certificate c to process p, $\mathcal{S}_{prop}$ remains silent, unless a delivery of c' to p has already occurred with $\mathsf{conflict}(c, c') = true$. In this case, $\mathcal{S}_{prop}$ triggers the notification

$$(\mathsf{detection}, \mathsf{sid}, \mathsf{ssid}, D, (c, c'))$$

where $\mathsf{Judge}(c, c')$ returns $\mathsf{dis}(D)$.

Let $\mathcal{E}_{prop} \in \mathfrak{E}_{prop}$. We define an environment $\mathcal{E}_{flood} \in \mathfrak{E}_{flood}$ that behaves exactly as $\mathcal{E}_{prop}$, except: propagation requests are replaced with corresponding flooding requests, and when $\mathcal{E}_{flood}$ receives a delivery notification for process p, it stores the certificate in $\mathsf{Cert}[p]$. If $\mathsf{Cert}[p]$ does not contain a pair of conflicting certificates, the environment behaves as if it received nothing.

Thus, we have:

$$\mathcal{E}_{flood} \mid \Pi_{\mathsf{ERFlood}} \mid \mathcal{A}_{flood} \simeq \mathcal{E}_{flood} \mid \mathcal{F}_{\text{flood}} \mid \mathcal{S}_{flood}.$$

Moreover, by construction:

$$\mathcal{E}_{flood} \mid \mathcal{F}_{\text{flood}} \mid \mathcal{S}_{flood} \simeq \mathcal{E}_{prop} \mid \mathcal{F}_{\text{prop}} \mid \mathcal{S}_{prop}.$$

and, as long as (*) $\mathcal{A}_{flood}$ delays the reception of messages in Delayed until $\mathcal{E}_{flood}$'s decision

$$\mathcal{E}_{flood} \mid \Pi_{\mathsf{ERFlood}} \mid \mathcal{A}_{flood} \simeq \mathcal{E}_{prop} \mid \mathcal{P}_{\text{prop}} \mid \mathcal{A}_{prop}$$

It follows that, if (*) holds with overwhelming probability:

$$\mathcal{E}_{prop} \mid \mathcal{P}_{\text{prop}} \mid \mathcal{A}_{prop} \simeq \mathcal{E}_{prop} \mid \mathcal{F}_{\text{prop}} \mid \mathcal{S}_{prop},$$

We now prove that (*) holds, except with negligible probability. We define $\mathcal{E}^*$ and $\mathcal{A}^*$ to behave identically to $\mathcal{E}_{flood}$ and $\mathcal{A}_{flood}$, except that every certificate c appearing on their external tape is replaced by c_i if $c \equiv c_i$ for some $i \in \{1, 2, 3\}$. Clearly, we have:

$$\mathcal{E}^* \mid \Pi_{\mathsf{ERFlood}} \mid \mathcal{A}^* \simeq \mathcal{E}_{flood} \mid \Pi_{\mathsf{ERFlood}} \mid \mathcal{A}_{flood}.$$

Now, by the specification of the flooding functionality, except with negligible probability, a correct process is expected to deliver a pair of conflicting certificates from $\{c_1, c_2, c_3\}$ *before* the delivery of all in-flight messages sent between so-far correct processes. Furthermore, the delayed messages in Delayed no longer affect the execution: a process never relays the same certificate twice. Thus, a correct process is expected to deliver a pair of conflicting certificates from $\{c_1, c_2, c_3\}$ before $\mathcal{A}_{flood}$ triggers delivery of the pending messages in Delayed. We can therefore conclude that, except with negligible probability, $\mathcal{A}_{flood}$ delays the reception of messages in Delayed until after $\mathcal{E}_{flood}$ has reached a decision. This concludes the proof.

□

B.3. The Accountable Confirmer Functionality $\mathcal{F}_{\text{conf}}^{\text{acc}} = \mathcal{F}_{\text{rat}} \triangleright \mathcal{F}_{\text{prop}}$

The accountable confirmer, introduced as the core of the $\mathcal{ABC}$ compiler in [2], [11], is defined in Functionality 3.

Functionality 3 $\mathcal{F}_{\text{conf}}^{\text{acc}}$

Parameters:
t: Optimistic Threshold
t_{acc}: Pessimistic Threshold
d: Number of dishonest processes that should be exposed in case of disagreement

Variables:
Set(Message) *Submitted* ▷ Set of submitted messages
Map(Message → Set(Process)) *Submissions* ▷ Maps each message m to the set of processes submitting m.

Participants:
$p_1, \ldots, p_n$: Parties interacting with the functionality.
Sim: Ideal adversary (simulator).
Judge: (public and fair) judge

Events:
request (submit, sid, ssid, m): A party submits a message m.
notification (confirm, sid, ssid, m): A party confirms the message m.

Functionality:
Upon receiving (submit, sid, ssid, m) from an uncorrupted process p_i:
Submitted $\leftarrow$ *Submitted* $\cup \{m\}$
Submissions$[m] \leftarrow$ *Submissions*$[m] \cup \{p_i\}$
Notify Sim of the submission of m by p_i

//conditions below are polled upon each new corruption and each new submit request

If $\exists m$ s.t. *Submitted* $= \{m\} \wedge |$*Submissions*$[m] \cap$ Correct$| \geq n - t$:
Send (confirm, sid, ssid, m) to all processes as a delayed output ▷ Conditioned Agreement and Liveness

If $\exists m, m'$ s.t. $m, m' \in$ *Submitted*, with $m \neq m'$: ▷ Liveness and Agreement are not guaranteed anymore
Optimistic Convergence is not guaranteed anymore: Sim can indefinitely delay any confirm notification.
For any $m'' \in$ *Submitted*, for any $p_i \in$ *Submissions*$[m'']$, Sim can choose at any time to forward (confirm, sid, ssid, m'') to p_i.

// Degraded Mode ($f \leq t_{acc}$)
Optimistic Convergence is not guaranteed anymore: Sim can indefinitely delay any confirm notification.
If two correct processes deliver (confirm, sid, ssid, m) and (confirm, sid, ssid, $m' \neq m$), respectively, then:
send (detection, sid, ssid, D_i, π_i) to each p_i as a delayed output, with $|D_i| \geq d$, such that:
the judge returns the (fair) verdict dis(D_i) on top of evidence π_i

The functionality $\mathcal{F}_{\text{conf}}^{\text{acc}}$ ensures:

- **Integrity**: A process cannot confirm a value it did not submit.
- **Optimistic Convergence**: If $f \leq t$ and all correct processes submit the same value v, they all eventually confirms v.
- **Accountability for agreement**: If two correct processes output different values, every correct process eventually obtains a proof π_i exposing d dishonest processes.

Accountability can be obtained from the conjunction of three distinct properties:

- **Strong-forensic support for agreement**: If two correct processes p_i and p_j confirm different values, they store 2 valid conflicting certificates c_i and c_j, where the underlying $\big((\text{valid}(\cdot), \text{conflict}(\cdot,\cdot)\big)$ predicates are propagation-friendly.
- **Accountability for conflict-freeness**: If two correct processes collectively propagate a pair of conflicting certificates, then every correct process p_i eventually obtains a proof π_i exposing d dishonest processes.
- **Involvement**: If a correct process confirms a value, it propagates the corresponding valid certificate.

Essentially, $\mathcal{F}_{\text{rat}}$ ensures (1) integrity, (2) optimistic convergence, and (3) forensic support for disagreement, while $\mathcal{F}_{\text{prop}}$ guarantees (4) accountability for conflict-freeness. The composition simply ensures (5) involvement.

This leads to a direct realization of $\mathcal{F}_{\text{conf}}^{\text{acc}}$ in the $(\mathcal{F}_{\text{rat}}, \mathcal{F}_{\text{prop}})$-hybrid model, presented in Algorithm 10.

Algorithm 10 $\mathcal{P}_{\text{conf}}^{\text{acc}} \leq \mathcal{F}_{\text{conf}}^{\text{acc}}$ on top of $\mathcal{F}_{\text{rat}}$ and $\mathcal{F}_{\text{prop}}$

Uses:
  $\mathcal{F}_{\text{rat}}$ ▷ Ratifier functionality for forensic support
  $\mathcal{F}_{\text{prop}}$ ▷ Propagator functionality for accountability

upon $(\mathsf{submit}, \mathsf{sid}, \mathsf{ssid}, v)$:
  Request $(\mathsf{submit}, \mathsf{sid}, \mathsf{ssid}, v)$ from $\mathcal{F}_{\text{rat}}$

upon receiving $(\mathsf{confirm}, \mathsf{sid}, \mathsf{ssid}, v, \mathit{cert})$ from $\mathcal{F}_{\text{rat}}$: ▷ Forensic support is guaranteed
  trigger $(\mathsf{confirm}, \mathsf{sid}, \mathsf{ssid}, v)$
  Request $(\mathsf{propagate}, \mathsf{sid}, \mathsf{ssid}, \mathit{cert})$ from $\mathcal{F}_{\text{prop}}$

upon receiving $(\mathsf{detection}, \mathsf{sid}, \mathsf{ssid}, D, \pi)$ from $\mathcal{F}_{\text{prop}}$:
  trigger $(\mathsf{detection}, \mathsf{sid}, \mathsf{ssid}, D, \pi)$ ▷ Accountability is achieved

Lemma 3. $\mathcal{P}_{\text{conf}}^{\text{acc}}$ (Algorithm 10) realizes $\mathcal{F}_{\text{conf}}^{\text{acc}}$ in the $(\mathcal{F}_{\text{rat}}, \mathcal{F}_{\text{prop}})$-hybrid model with no additional complexity overhead beyond that of the realizations of $\mathcal{F}_{\text{rat}}$ and $\mathcal{F}_{\text{prop}}$.

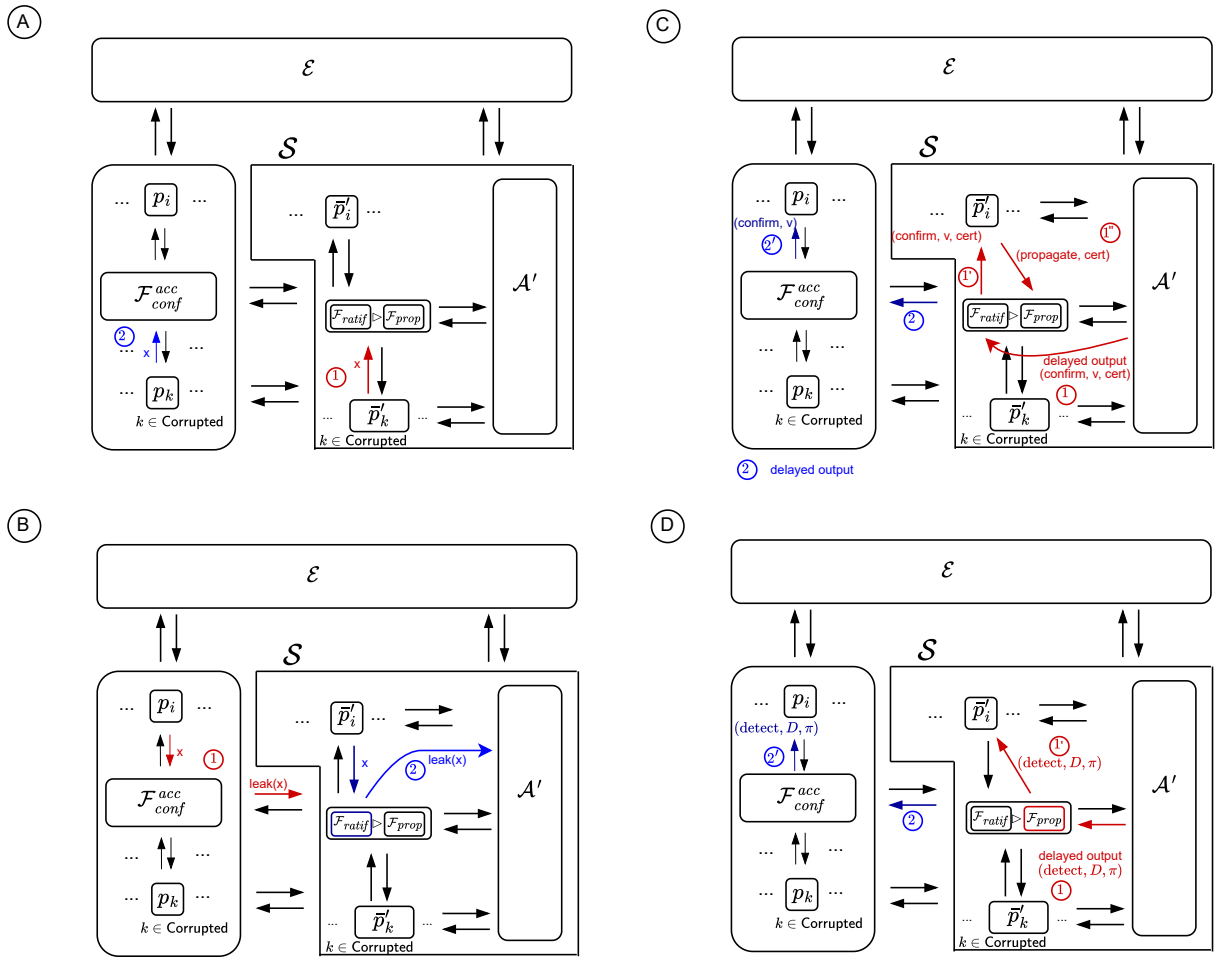

Figure 2: $\mathcal{F}_{\text{conf}}^{\text{acc}}$'s realization

PROOF.

Foreword. We consider the ideal protocol

$$\mathcal{F}_{\text{conf}}^{\text{acc}} = p_1 \mid \cdots \mid p_n \mid \mathcal{F}_{\text{conf}}^{\tilde{\text{acc}}}$$

where $p_1, \ldots, p_n$ represent the dummy processes, and its realization in the $(\mathcal{F}_{\text{rat}}, \mathcal{F}_{\text{prop}})$-hybrid model

$$\mathcal{P}_{\text{conf}}^{\text{acc}} = \bar{p}_1 \mid \cdots \mid \bar{p}_n \mid \mathcal{F}_{\text{rat}} \mid \mathcal{F}_{\text{prop}}.$$

Let $\mathcal{A}$ be the adversary of $\mathcal{P}_{\text{conf}}^{\text{acc}}$. The simulator $\mathcal{S}$ internally simulates the adversary, denoted by $\mathcal{A}'$, as well as the ratifier $\mathcal{F}_{\text{rat}}$ and the propagator $\mathcal{F}_{\text{prop}}$. Additionally, for *each* $k \in [1:n]$, it simulates $\bar{p}_k$, denoted as $\bar{p}'_k$. Let us note it can do so, since no information is private in the realization.

Messages received by $\mathcal{S}$ from the environment $\mathcal{E}$ are forwarded to $\mathcal{A}'$. Likewise, any messages sent by $\mathcal{A}'$ that are destined for the environment are forwarded to $\mathcal{E}$. The simulator $\mathcal{S}$ keeps the corruption status synchronized; that is, whenever $\mathcal{A}$ corrupts a process $\bar{p}'_k$, the simulator also corrupts the corresponding process p_k.

Incoming messages from a corrupted process p_k to $\mathcal{S}$ are forwarded to $\mathcal{A}'$ on behalf of the corresponding corrupted process $\bar{p}'_k$. Conversely, whenever a corrupted process $\bar{p}'_k$ attempts to output a message to the environment, $\mathcal{S}$ instructs the corresponding corrupted process p_k to output the same message to $\mathcal{E}$.

$\mathcal{F}_{\text{conf}}^{\text{acc}} = (\mathcal{F}_{\text{rat}} \rhd \mathcal{F}_{\text{prop}})$**.** We now describe the strategy of the simulator, beyond the foreword. The process consists of four situations, labeled A, B, C, and D, depicted in Figure 2.

- **A**: Whenever a malicious process $\bar{p}'_k$ attempts to send a request to $\mathcal{F}_{\text{rat}}$, the simulator $\mathcal{S}$ instructs p_k to send the same request to $\mathcal{F}_{\text{conf}}^{\text{acc}}$.
- **B**: When a submit request is sent by any process p_j (malicious or correct), the simulator $\mathcal{S}$ is immediately notified via the corresponding leak tape. Upon notification, $\mathcal{S}$ simulates $\bar{p}'_j$ to ensure it writes the same submit request to $\mathcal{F}_{\text{rat}}$, thereby triggering the corresponding notification received by $\mathcal{A}'$ on the corresponding leak tape.
- **C**: Whenever $\mathcal{A}'$ orders the forwarding of a (delayed) confirm notification from $\mathcal{F}_{\text{rat}}$ to $\bar{p}_j$, the simulator $\mathcal{S}$ writes the corresponding order on the appropriate tape of $\mathcal{F}_{\text{rat}}$. This triggers the reception of the confirm notification by $\bar{p}'_j$, which then sends a propagate request to $\mathcal{F}_{\text{prop}}$. Subsequently, $\mathcal{S}$ orders the forwarding of the same (delayed) confirm notification from $\mathcal{F}_{\text{conf}}^{\text{acc}}$ to p_j, which then receives it.
- **D**: Whenever $\mathcal{A}'$ orders $\mathcal{F}_{\text{prop}}$ to forward the delayed output $(\mathsf{detection}, \mathsf{sid}, \mathsf{ssid}, D, \pi)$ to process $\bar{p}_j$, process $\bar{p}'_j$ receives it. The simulator $\mathcal{S}$ then instructs $\mathcal{F}_{\text{conf}}^{\text{acc}}$ to forward the delayed output $(\mathsf{detection}, \mathsf{sid}, \mathsf{ssid}, D, \pi)$ to process p_j, which receives it as well.

Nominal Case. In this case, the adversary does not corrupt more than t processes. Correct processes are expected to unanimously submit the same value v to $\mathcal{F}_{\text{conf}}^{\text{acc}}$, and accordingly to $\mathcal{F}_{\text{rat}}$ in the simulation. Furthermore, the functionality $\mathcal{F}_{\text{rat}}$ ensures Optimistic Convergence. This guarantees that situation C will occur, resulting in all processes confirming v in both worlds.

Degraded Case. In this case, the adversary can corrupt up to t_{acc} processes. The adversary may attempt to violate both the optimistic convergence and agreement properties of $\mathcal{F}_{\text{rat}}$. Violating liveness is not problematic in the degraded mode. However, if agreement is violated, the forensic support property ensures that the adversary $\mathcal{A}'$ must accompany conflicting confirm notifications with conflicting certificates. Consequently, a pair of correct processes will collectively send a pair of conflicting certificates to $\mathcal{F}_{\text{prop}}$. By the accountability property for conflicting certificates, the adversary $\mathcal{A}'$ will eventually have to trigger the delayed output for detection. This action will also be performed by the simulator, thereby ensuring accountability for disagreement. □

B.4. The $\mathcal{ABC}$ Transformation: $\mathcal{F}_{\text{ea}}^{\text{acc}} = \mathcal{ABC}(\mathcal{F}_{\text{ea}}) = \mathcal{F}_{\text{ea}} \rhd \mathcal{F}_{\text{conf}}^{\text{acc}}$

We are now ready to prove the correctness of the $\mathcal{ABC}$ compiler (see Algorithm 11) in the $\mathcal{F}_{\text{conf}}^{\text{acc}}$-hybrid model. The transformation sequentially composes any easily accountable functionality $\mathcal{F}_{\text{ea}}$ with the accountable confirmer $\mathcal{F}_{\text{conf}}^{\text{acc}}$ to obtain its accountable counterpart $\mathcal{F}_{\text{ea}}^{\text{acc}}$.

Lemma 4. Let $\mathcal{F}_{\text{ea}}$ be an easily accountable functionality with t'-resiliency, and let $\mathcal{F}_{\text{conf}}^{\text{acc}}$ be the accountable confirmer functionality parametrized with (t, t_{acc}, d). Define $\mathcal{P}_{\text{ea}}^{\text{acc}} = \mathcal{ABC}(\mathcal{F}_{\text{ea}}, \mathcal{F}_{\text{conf}}^{\text{acc}})$ (see Algorithm 11). Then, $\mathcal{P}_{\text{ea}}^{\text{acc}}$ realizes $\mathcal{F}_{\text{ea}}$ with resiliency $\min(t, t')$. Moreover, if $\min(t, t') < f \leq t_{acc}$ and a disagreement occurs, every correct process eventually obtains an externally verifiable proof that exposes at least d dishonest processes.

Furthermore, the message, communication, and round complexity of $\mathcal{P}_{\text{ea}}^{\text{acc}}$ incur no additional overhead beyond the sum of the corresponding complexity metrics of the individual realizations of $\mathcal{F}_{\text{ea}}$ and $\mathcal{F}_{\text{conf}}^{\text{acc}}$.

PROOF.

Algorithm 11 $\mathcal{ABC}(\mathcal{F}_{\text{ea}}) \leq \mathcal{F}_{\text{ea}}^{\text{acc}}$ on top of $\mathcal{F}_{\text{conf}}^{\text{acc}}$

Uses:
  $\mathcal{F}_{\text{ea}}$ ▷ An easily accountable functionality in the sense of Definition 3
  $\mathcal{F}_{\text{conf}}^{\text{acc}}$ ▷ Accountable confirmer functionality (see Functionality 3)

Local Variables:
  Value $predecided_i \leftarrow \perp$

upon $(\mathsf{propose}, \mathsf{sid}, \mathsf{ssid}, v)$:
  Request $(\mathsf{propose}, \mathsf{sid}, \mathsf{ssid}, v)$ from $\mathcal{F}_{\text{ea}}$

upon receiving $(\mathsf{decide}, \mathsf{sid}, \mathsf{ssid}, w)$ from $\mathcal{F}_{\text{ea}}$: ▷ Forensic support is guaranteed
  $predecided_i \leftarrow w$
  Request $(\mathsf{submit}, \mathsf{sid}, \mathsf{ssid}, w)$ from $\mathcal{F}_{\text{conf}}^{\text{acc}}$

upon receiving $(\mathsf{confirm}, \mathsf{sid}, \mathsf{ssid}, predecided_i)$ from $\mathcal{F}_{\text{conf}}^{\text{acc}}$:
  trigger $(\mathsf{decide}, \mathsf{sid}, \mathsf{ssid}, predecided_i)$ from $\mathcal{F}_{\text{conf}}^{\text{acc}}$ ▷ Agreement and Liveness are preserved

upon receiving $(\mathsf{detection}, \mathsf{sid}, \mathsf{ssid}, D, \pi)$ from $\mathcal{F}_{\text{conf}}^{\text{acc}}$:
  trigger $(\mathsf{detection}, \mathsf{sid}, \mathsf{ssid}, D, \pi)$ ▷ Accountability for agreement is achieved

Foreword. We consider the ideal protocol

$$\mathcal{F}_{\text{ea}}^{\text{acc}} = p_1 \mid \cdots \mid p_n \mid \tilde{\mathcal{F}_{\text{ea}}^{\text{acc}}}$$

where $p_1, \ldots, p_n$ represent the dummy processes, and its realization in the $(\mathcal{F}_{\text{ea}}, \mathcal{F}_{\text{conf}}^{\text{acc}})$-hybrid model

$$\mathcal{P}_{\text{ea}}^{\text{acc}} = \bar{p}_1 \mid \cdots \mid \bar{p}_n \mid \mathcal{ABC}(\mathcal{F}_{\text{ea}}, \mathcal{F}_{\text{conf}}^{\text{acc}}) = (\check{p}_1 \mid \mathring{p}_1) \mid \cdots \mid (\check{p}_n \mid \mathring{p}_n) \mid \mathcal{F}_{\text{ea}} \mid \mathcal{F}_{\text{conf}}^{\text{acc}},$$

where $\check{p}_i$ corresponds to the same machine as p_i and $\mathring{p}_i$ corresponds to machine that (1) upon receiving a notification $(\mathsf{decide}, *, y)$ from $\mathcal{F}_{\text{ea}}$, $\mathring{p}_i$ sends a request $(\mathsf{submit}, *, y)$ to $\mathcal{F}_{\text{conf}}^{\text{acc}}$, and triggers the decision of y, upon receiving a notification $(\mathsf{confirm}, *, y)$ from $\mathcal{F}_{\text{conf}}^{\text{acc}}$.

Let $\mathcal{A}$ be the adversary of $\mathcal{P}_{\text{ea}}^{\text{acc}}$. The simulator $\mathcal{S}$ internally simulates the adversary, denoted by $\mathcal{A}'$, and the accountable confirmer $\mathcal{F}_{\text{conf}}^{\text{acc}}$. Additionally, for each $k \in [1:n]$, $\mathcal{S}$ simulates $\mathring{p}_k$, denoted $\mathring{p}'_k$ and, upon the corruption of process p_k, it simulates a copy of the corresponding process of the compiled protocol, denoted $\bar{p}'_k$. Importantly, $\mathcal{S}$ does not simulate $\mathcal{F}_{\text{ea}}$, which might manage private information, such as the prover's witness in the one-to-many zkPoK functionality [19].

Messages received by $\mathcal{S}$ from the environment $\mathcal{E}$ are forwarded to $\mathcal{A}'$. Similarly, any messages sent by $\mathcal{A}'$ that are destined for the environment are forwarded to $\mathcal{E}$. The simulator $\mathcal{S}$ keeps the corruption status synchronized; that is, whenever $\mathcal{A}$ corrupts a process $\bar{p}'_k$, the simulator also corrupts the corresponding process p_k, learns its state—including its cryptographic material.

Incoming messages from a corrupted process p_k to $\mathcal{S}$ are forwarded to $\mathcal{A}'$ on behalf of the corresponding corrupted process $\bar{p}'_k$. Conversely, whenever a corrupted process $\bar{p}'_k$ attempts to output a message to the environment, $\mathcal{S}$ instructs the corresponding corrupted process p_k to output the same message to $\mathcal{E}$.

The $\mathcal{ABC}$ Emulation. Let us describe the simulator strategy, beyond the foreword. We describe 6 situations A, B, C, D, E, and F, depicted in Figure 3.

- **A**: Whenever a malicious process $\bar{p}'_k$ intends to send a request to $\mathcal{F}_{\text{ea}}$, the simulator $\mathcal{S}$ instructs p_k to send the same request to $\mathcal{F}_{\text{ea}}^{\text{acc}}$.
- **B**: When a request is sent by a correct process, the simulator $\mathcal{S}$ is immediately notified via the corresponding leak tape. Upon notification, $\mathcal{S}$ writes the notification on the leak tape of $\mathcal{A}'$, related to $\mathcal{F}_{\text{ea}}$. Let us note that the leak does not necessarily include the request's payload. For example, if $\mathcal{F}_{\text{ea}}$ represents the one-to-many zkPoK functionality [19], the prover's witness is not leaked.
- **C**: Whenever $\mathcal{A}'$ orders to forward a (delayed) notification from $\mathcal{F}_{\text{ea}}$, the simulator orders the same to $\mathcal{F}_{\text{ea}}^{\text{acc}}$, except for the special decide notification (see D).
- **D**: Whenever $\mathcal{A}'$ orders $\mathcal{F}_{\text{ea}}$ to forward the delayed output $(\mathsf{decide}, \mathsf{sid}, \mathsf{ssid}, y)$ to process $\bar{p}_j$, the simulator writes the notification on the correponding tape of $\mathring{p}'_j$. Then, situation E occurs immediately after.
- **E**: Upon receiving $(\mathsf{decide}, \mathsf{sid}, \mathsf{ssid}, y)$, $\mathring{p}'_j$ reacts by sending $(\mathsf{submit}, \mathsf{sid}, \mathsf{ssid}, y)$ to $\mathcal{F}_{\text{conf}}^{\text{acc}}$.
- **F**: Whenever $\mathcal{A}'$ orders $\mathcal{F}_{\text{conf}}^{\text{acc}}$ to forward the delayed output $(\mathsf{confirm}, \mathsf{sid}, \mathsf{ssid}, y)$ or the delayed output $(\mathsf{detection}, \mathsf{sid}, \mathsf{ssid}, D, \pi)$ to process $\bar{p}_j$, $\mathcal{F}_{\text{conf}}^{\text{acc}}$ does so, while the simulator orders $\mathcal{F}_{\text{ea}}^{\text{acc}}$ to do the same, which it does as well.

Nominal Case. In this case, the adversary does not corrupt more than t processes. Moreover, the functionality $\mathcal{F}_{\text{ea}}$ is supposed to ensure Agreement & Termination. This means that the succession of situations D and E will occur for a subset

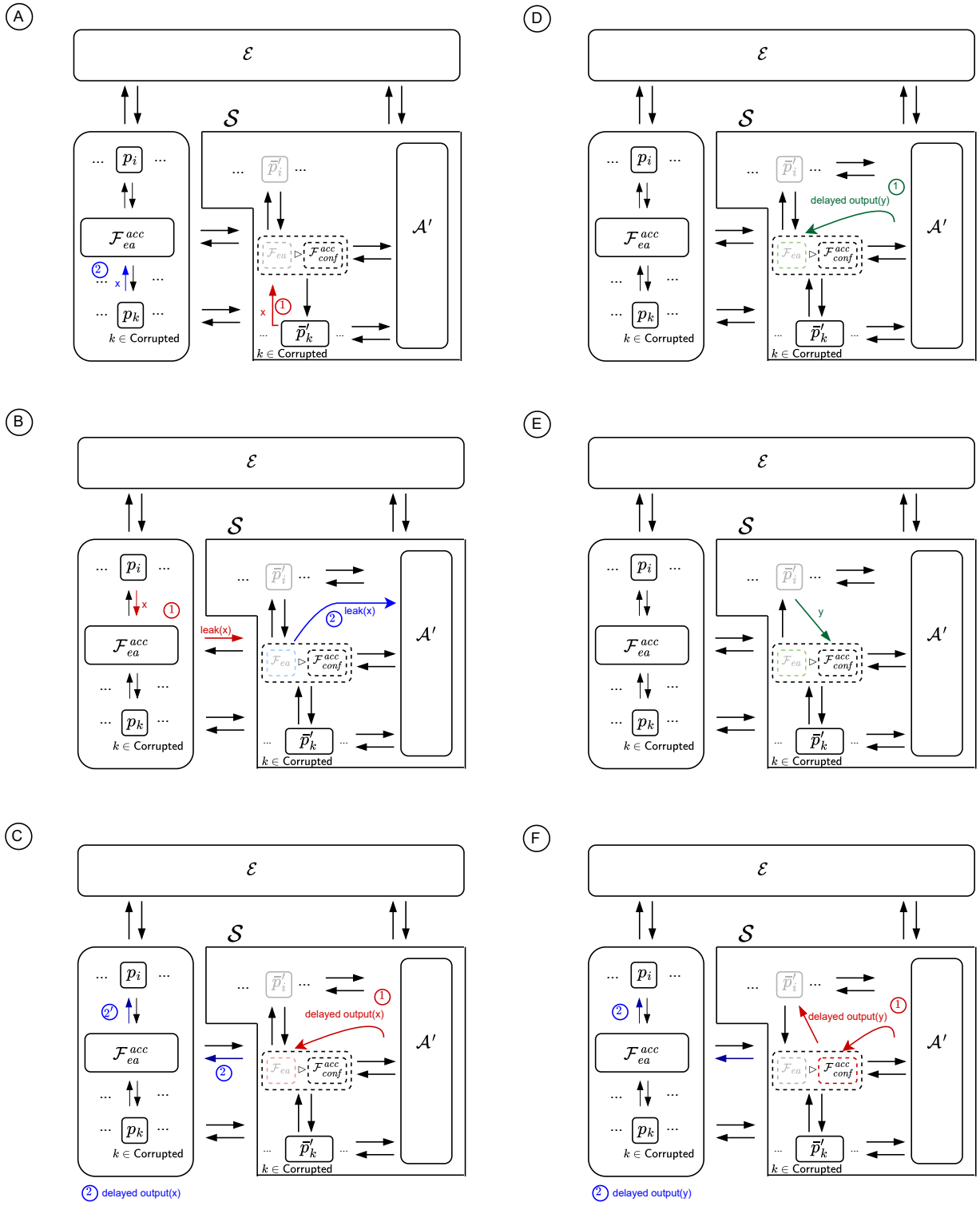

Figure 3: UC emulation of the $\mathcal{ABC}$ transformation

of processes $\mathring{p}'_*$. Integrity guarantees the preservation of Agreement property. Assume all processes $\mathring{p}'_*$ face the succession of situations D and E. By agreement, they all submit the same value, say y. By optimistic convergence, situation F must occur, meaning they all confirm y. At the same time, all correct processes p_* do the same. This means that if Termination corresponds to Totality, Termination is preserved. Now, assume the opposite. This means that Totality is not ensured. Thus, Termination corresponds Partial Decidability. Hence, it is allowed for any subset of processes not to decide. The remaining structure of the simulation ensures that the ideal and the real world are indistinguishable.

Degraded Case. In this case, the adversary can corrupt up to t_{acc} processes. The adversary can tackle both the optimistic convergence and the agreement property of $\mathcal{F}_{\text{conf}}^{\text{acc}}$. Violating liveness is not an issue in the degraded mode. If agreement is violated, the accountability property implies that the adversary $\mathcal{A}'$ will have to eventually trigger the delayed output for detection. This will be performed by the simulator as well, ensuring accountability.

□

Consequently, Theorem 5 holds.

Theorem 5. Let J_{rat} be the judge defined in Algorithm 3, and $\mathcal{F}_{\text{ea}}^{\text{acc}}$ be the $(t_{acc}, B_{\delta,\hat{\delta}}^{\epsilon,\lambda}, J_{rat})$-accountable counterpart of an easily accountable agreement functionality $\mathcal{F}_{\text{ea}}$ (as defined in Definition 3). If $\mathcal{P}$ UC-realizes $\mathcal{F}_{\text{ea}}$, then $\mathcal{ABC}^{++}(\mathcal{P})$ UC-realizes $\mathcal{F}_{\text{ea}}^{\text{acc}}$ with complexity overhead described in Theorem 4.

PROOF. By conjunction of Lemmas 1 to 4 and composability of UC-realization. □

Appendix C. Verifiable Committee Sampling

This section elaborates on the Verifiable Committee Sampling (VCS) scheme. First, we prove Theorem 1. Second, we present the corresponding ideal functionality $\mathcal{F}_{\text{vcs}}$.

C.1. Committee-Sampling via Verifiable Random Function

Assuming the Random Oracle Model [117], a VRF [16] can be instantiated from any digital signature scheme with the *uniqueness* property, where it is computationally infeasible to produce two distinct valid signatures for the same message and public key. The BLS signature scheme [95], [118] satisfies this property while also supporting aggregation [87], [119] and resisting rogue-key attacks [88]. To construct a VRF, a process simply signs a step-specific label and hashes the resulting signature. Any verifier can then check the signature and recompute the hash locally. This construction yields a non-interactive lottery functionality: a process is elected if the hash of its signature falls below a predefined threshold. To ensure unpredictability and fairness, public keys must be published on a public bulletin-board-style PKI (see §D.2), and the label used for signing must be concatenated with a common entropy source seed. As shown by Rambaud [48], seed can be instantiated as the hash of the public keys themselves, at the cost of introducing a confined bias that does not affect the forensic guarantees with more than a negligible probability. Thanks to the aggregation-friendliness of BLS, it was possible to realize a concretely efficient aggregate VCS functionality (called non-interactive lottery in [17]), where proofs of eligibility from multiple processes can be combined into just two group elements and verified using a single pairing check.

Theorem 1. Let $\mathsf{H}(s,\lambda)$ denote the random variable that returns the set of (so-far honest) processes p_i such that the sampling event $\mathsf{sample}_i(s,\lambda)$ is not preceded by the corruption of p_i in the execution, and let $C(s,\lambda) = \{p_i \in \Psi | \mathsf{sample}_i(s,\lambda) = \langle true, * \rangle\}$, sampled uniformly at random with expected size λ. [5]

- **Liveness.** With $W = (1-\delta)(\frac{2}{3}+\epsilon)\lambda$, we have:

$$\Pr\left[|C(s,\lambda) \cap \mathsf{H}(s,\lambda)| \leq W\right] \leq \exp\left(-\frac{\delta^2(2+3\epsilon)}{6}\lambda\right)$$

- **Forensics.** Let $Q, Q' \subseteq C(s,\lambda)$ with $|Q| = |Q'| = W_\delta^{\epsilon,\lambda}$. Let $B = 2W_\delta^{\epsilon,\lambda} - (1+\hat{\delta})\lambda$ Then:

$$\Pr\left[|Q \cap Q'| \leq B\right] \leq \exp\left(-\frac{\hat{\delta}^2}{2+\hat{\delta}}\lambda\right)$$

PROOF. Let X_j be the Bernoulli random variable that indicates if $p_j \in C(s,\lambda)$, so $\Pr[X_j = 1] = \lambda/n$. Define $Z_L = \sum_{p_j \in \mathsf{H}} X_j$ and $Z_F = \sum_{p_j \in \Pi} X_j$. By linearity of expectation, $\mathbb{E}(Z_L) \geq (\frac{2}{3}+\epsilon)\lambda$ and $\mathbb{E}(Z_F) = \lambda$.

- (Liveness) $|C(s,\lambda) \cap \mathsf{H}(s,\lambda)| \sim Z_L$. By Chernoff bound, $\Pr[Z_L \leq (1-\delta)\cdot(\frac{2}{3}+\epsilon)\lambda] \leq \exp\left(-\frac{\delta^2(2+3\epsilon)}{6}\lambda\right)$

5. Here, Pr represents the worst-case probability over adversarial strategies with up to t adaptive corruptions (no liveness guarantees when $f > t$).

- (Forensic) $|Q \cap Q'| \geq |Q| + |Q'| - |Q \cup Q'| \geq 2W_\delta - Z_F$.
  By Chernoff bound:
  $\Pr[Z_F \geq (1+\hat{\delta}) \cdot \lambda] \leq \exp\left(-\frac{\hat{\delta}^2}{2+\hat{\delta}}\lambda\right)$
  $\Pr[2W_\delta - Z_F \leq 2W_\delta - (1+\hat{\delta}) \cdot \lambda] \leq \exp\left(-\frac{\hat{\delta}^2}{2+\hat{\delta}}\lambda\right)$
  $\Pr[|Q \cap Q'| \leq B_{\delta,\hat{\delta}}] \leq \exp\left(-\frac{\hat{\delta}^2}{2+\hat{\delta}}\lambda\right)$

□

C.2. The ideal functionality $\mathcal{F}_{\text{vcs}}$

The functionality $\mathcal{F}_{\text{vcs}}$, presented in Functionality 4, allows processes to elect committees without communication and later prove their election, but differs from $\mathcal{F}_{\text{mine}}$ [50, Fig. 2] and $\mathcal{F}_{eligib}$ [48, Fig. 1] in that proofs are explicitly part of the specification, rather than being implicit as in $\mathcal{F}_{\text{mine}}$ and $\mathcal{F}_{eligib}$. This approach highlights the non-interactive nature of the scheme. The reader can refer to Rambaud's presentation [48] for the UC-realization of the corresponding *eligibility* functionality, built on top of a (synchronous) *bulletin-board* PKI functionality, where the need of a delayed public random beacon can be removed at the cost of a confined bias that does not affect the forensic guarantees with more than a negligible probability.

Functionality 4 $\mathcal{F}_{\text{vcs}}$: Verifiable Committee Sampling

Parameters:
$\lambda \in \mathbb{N}$: A statistical parameter: the expected value of the underlying Bernoulli random variable

Participants:
$p_1, \ldots, p_n$: Parties interacting with the functionality.
Sim: Ideal process adversary (simulator).

Local State:
T: A table of eligibility records, initially empty.

Events:
request (eligibility, sid, ssid): A party queries whether it is eligible to participate in subsession ssid.
request (verify, sid, ssid, p_j, π): A party or verifier checks the p_j's eligibility proof π for subsession ssid.

Functionality:
Upon receiving (eligibility, sid, ssid) from p_i for the first time:
  Toss a coin $coin_i \sim \text{Bernoulli}(\lambda/n)$.
    Generate a proof π_i
    Store (sid, ssid, $p_i, coin_i, \pi_i$) in T.
    Send (eligibility, sid, ssid, $coin_i, \pi_i, p_i$) to Sim.
    Send (eligibility, sid, ssid, $coin_i, \pi_i$) to p_i.

Upon receiving (verify, sid, ssid, p_i, π_i) from any process:
  Let $b :=$ true if (sid, ssid, $p_i, 1, \pi_i$) $\in$ T, else $b :=$ false
  Return (verification, sid, ssid, p_i, b)

Appendix D. Multi-Signature

This section elaborates on the multi-signature scheme. First, we provide a game-based definition. Second, we present the corresponding ideal functionality $\mathcal{F}_{\text{msig}}$.

D.1. Game-based Definition

In this section, we present the syntax and security definitions for dynamic accountable non-interactive multi-signature schemes. We follow the presentation of [120], that we adapt to multi-signature, following [90].

Definition 4 (Dynamic Accountable Non-Interactive Multi-Signature). A dynamic accountable non-interactive multi-signature scheme DANMSS is a tuple of polynomial-time algorithms DANMSS $= (\text{setup}, \text{kgen}, \text{sign}, \text{cb}, \text{ver})$ defined as follows:

1) setup$(1^\kappa) \to$ pp: The setup algorithm takes as input a security parameter κ and outputs public parameters pp (which are implicitly given as input to all other algorithms).

2) $\mathsf{kg}() \to (\mathsf{pk}_i, \mathsf{sk}_i)$: Every signer (process) can generate a key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$[6]
3) $\mathsf{sign}(\mathsf{sk}_i, m) \to \sigma_i$: The signing algorithm takes as input a secret key share sk_i and a message $m \in \{0,1\}^{poly(\kappa)}$. It outputs a signature σ_i.
4) $\mathsf{cb}(m, \{(\mathsf{pk}_i, \sigma_i)\}_{i \in G}) \to \sigma/\perp$: The combination algorithm takes as input a message m, and a set of tuples $(\mathsf{pk}_i, \sigma_i)$ consisting of public keys and individual signatures from a certain group of signers G. It outputs either a signature σ or $\perp$.
5) $\mathsf{verif}(m, \sigma, \{\mathsf{pk}_i\}_{i \in G}) \to \{0,1\}$: The verification algorithm takes as input a message m, a signature σ, and a set of public keys $\{\mathsf{pk}_i\}_{i \in G}$. It outputs 1 (accept) or 0 (reject).

Furthermore, the scheme satisfies the properties of *Unforgeability* and *Robustness*, that is, it is both UF-CMA secure and RB-CMA secure, as defined below.

Unforgeability Security Game (UF-CMA$^{\mathcal{A}}_{\mathsf{MS}}$)

1) $\mathsf{pp} \leftarrow \mathsf{setup}(1^\kappa)$
2) $\{\mathsf{pk}_i, \mathsf{sk}_i\}_{i \in [n]} \leftarrow \mathsf{kg}(\mathsf{pp})$
3) Initialize $\mathcal{H} := \Psi$, and for each message m, $Q[m] := \emptyset$
4) Input: $(\mathsf{pp}, \{\mathsf{pk}_i\}_{i \in [n]})$
5) $(m, \sigma, \mathcal{I}) \leftarrow \mathcal{A}^{\mathsf{Corr},\mathsf{Sign}}(\mathsf{pp}, \{\mathsf{pk}_i\}_{i \in [n]})$
6) If $\mathcal{I} \cap (\mathcal{H} \setminus \mathcal{Q}[m]) \neq \emptyset$ and $\mathsf{verif}(m, \sigma, \{\mathsf{pk}_i\}_{p_i \in \mathcal{I}}) = 1$, return 1
7) Return 0

Oracle $\mathsf{Corr}(p_i)$**:**

$\mathcal{H} := \mathcal{H} \setminus \{p_i\}$
Return sk_i

Oracle $\mathsf{Sign}(p_i, m)$**:**

$Q[m] \leftarrow Q[m] \cup \{p_i\}$
Return $\mathsf{sign}(m, \mathsf{sk}_i)$

Robustness Security Game (RB-CMA$^{\mathcal{A}}_{\mathsf{MS}}$)

1) $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\kappa)$
2) $\{\mathsf{pk}_i, \mathsf{sk}_i\}_{i \in [n]} \leftarrow \mathsf{kg}(\mathsf{pp})$
3) Initialize $\mathcal{H} := \Psi$, and for each message m, $Q[m] := \emptyset$
4) Input: $(\mathsf{pp}, \{\mathsf{pk}_i\}_{i \in [n]})$

// Verification of honest partial signatures are always successful

5) $(i, m) \leftarrow \mathcal{A}^{\mathsf{Corr},\mathsf{Sign}}(\mathsf{pp}, \{\mathsf{pk}_i\}_{i \in [n]})$
6) $\sigma_i \leftarrow \mathsf{sign}(m, \mathsf{sk}_i)$
7) If $\mathsf{verif}(m, \sigma_i, \mathsf{pk}_i) \neq 1$, return 1

// Combining valid individual signatures must yield valid multi-signatures

8) $(G, m', \{\sigma_i\}_{p_i \in G}) \leftarrow \mathcal{A}^{\mathsf{Corr},\mathsf{Sign}}(\mathsf{pp}, \{\mathsf{pk}_i\}_{i \in [n]})$
9) If $\mathsf{verif}(m', \sigma_i, \mathsf{pk}_i) \neq 1$ for some $p_i \in G$, return 1
10) $\sigma \leftarrow \mathsf{cb}(m', \{(\mathsf{pk}_i, \sigma_i)\}_{p_i \in G})$
11) If $\mathsf{verif}(m', \sigma, \{\mathsf{pk}_i\}_{p_i \in G}) \neq 1$, return 1
12) Return 0

Figure 4: The unforgeability security game UF-CMA$^{\mathcal{A}}_{\mathsf{MSS}}$ and the robustness security game RB-CMA$^{\mathcal{A}}_{\mathsf{MSS}}$ for a multi-signature scheme

Unforgeability.. The unforgeability property is defined using the **UF-CMA**$^{\mathcal{A}}_{\mathsf{MS}}$ game (see Figure 4). Let $\mathcal{A}$ be an adversary in the **UF-CMA**$^{\mathcal{A}}_{\mathsf{MS}}$ game. The adversary receives as input the public parameters pp, and the individual public keys $\{\mathsf{pk}_i\}_{i \in [n]}$.

At any point, $\mathcal{A}$ can request individual signatures on a message m from any honest signer p_i by querying the oracle $\mathsf{Sign}(p_i, m)$. The game maintains a list $\mathsf{Q}[m]$ that records the subset of signers queried by $\mathcal{A}$ for individual signatures on m. Initially, $\mathsf{Q}[m] = \emptyset$ for all m.

6. Later, each process should be bind to a unique public key by a certification authority (CA) functionality $\mathcal{F}_{CA}$, whose realization is discussed below. Additionally, the CA should prevent rogue-key attacks, where a set of corrupted signers maliciously choose their public keys as functions of honestly generated keys. The scheme of [88] is resistant to such a rogue-key attack, thus avoiding classic defense requiring that users prove knowledge (or possession [91]) of the secret key during public key registration, a setting known as the knowledge of secret key (KOSK) assumption.

The adversary can corrupt any signer using the Corr oracle. Upon corrupting any party $p_i \in \Psi$, $\mathcal{A}$ learns its signing key sk_i. Since the internal state used in all individual signings is efficiently computable from the signing key and public messages, revealing only the signing key suffices upon corruption.

Finally, $\mathcal{A}$ wins the game if it outputs a valid forgery $(m^*, \sigma^*, \mathcal{I})$ such that there exists an uncorrupted process $p_i \in \mathcal{I}$, whose individual signature has not been queried for m^*, and is made accountable for σ^*.

Definition 5 (Unforgeability Under Chosen Message Attack). Let $\mathsf{DANMSS} = (\text{setup}, \text{kgen}, \text{sign}, \text{cb}, \text{ver})$ be a (dynamic accountable non-interactive) multi-signature scheme. We say that DANMSS is **UF-CMA secure** if, for all PPT adversaries $\mathcal{A}$:

$$\Pr[\mathsf{UF\text{-}CMA}^{\mathcal{A}}_{\mathsf{MS}}(\kappa) \Rightarrow 1] = \mathsf{negl}(\kappa).$$

Robustness.. The robustness property is formalized using the **RB-CMA**$^{\mathcal{A}}_{MS}$ game (see Figure 4). The robustness property ensures that the protocol behaves as expected for honest parties. More specifically, it guarantees:

1) The verification algorithm verif should always accept individual signatures generated by honest signers.
2) If valid individual signatures (accepted by verif) are combined using the cb algorithm, the resulting multi-signature should be accepted by verif, except with negligible probability.

This property ensures that maliciously generated individual signatures cannot prevent an honest aggregator from efficiently computing a valid multi signature.

Definition 6 (Robustness Under Chosen Message Attack). Let $\mathsf{DANMSS} = (\text{setup}, \text{kgen}, \text{sign}, \text{cb}, \text{ver})$ be a (dynamic accountable non-interactive) multi-signature scheme. We say that DANMSS is **RB-CMA secure** if, for all PPT adversaries $\mathcal{A}$:

$$\Pr[\mathsf{RB\text{-}CMA}^{\mathcal{A}}_{\mathsf{MS}}(\kappa) \Rightarrow 1] = \mathsf{negl}(\kappa).$$

D.2. The Ideal Functionality $\mathcal{F}_{\text{msig}}$

The ideal multi-signature functionality, denoted $\mathcal{F}_{\text{msig}}$, is specified in Functionality 5 and extends the standard (individual) signature functionality $\mathcal{F}_{\text{sig}}$ from [92] (an updated version of [92]). The key enhancements introduced are: (1) each signature is explicitly associated with a set of public keys, representing a group of signers rather than a single entity, and (2) multiple signatures can be combined into a single multi-signature, provided that each individual signature successfully verifies. We conjecture that, by an argument similar to the proof of [92, Theorem 2], a DANMSS satisfying unforgeability and robustness under chosen-message attacks can be used to UC-realize $\mathcal{F}_{\text{msig}}$.

Appendix E.
The Certification Authority $\mathcal{F}_{\text{CA}}$

We assume the existence of a certification authority functionality, denoted $\mathcal{F}_{\text{CA}}$ (see Functionality 6), responsible for registering party identities along with their corresponding public keys [92]. A distributed realization of $\mathcal{F}_{\text{CA}}$ requires at least $2/3$ correct processes [54], and adopting an accountable variant would introduce a circular dependency. However, in the unauthenticated synchronous setting, $\mathcal{F}_{\text{CA}}$ can be emulated even in the presence of an unbounded number of Byzantine processes, provided that digital signatures are available and the adversary's hash rate is computationally bounded [121], [122]. This allows for a practical two-phase approach:

- Setup Phase: Establish an ideal public key infrastructure (PKI) under a conservative bound Δ on message delay.
- Main Phase: Execute asynchronous protocols atop the established PKI, avoiding the inherent trade-off between safety and efficiency imposed by Δ.

The certification functionality $\mathcal{F}_{\text{cert}}$, which ensures a direct binding between messages and the signer's identity, as well as the message authentication functionality $\mathcal{F}_{\text{auth}}$, can both be securely realized in the $(\mathcal{F}_{(\text{m})\text{sig}}, \mathcal{F}_{\text{CA}})$-hybrid model [92].

Functionality 5 $\mathcal{F}_{\mathsf{msig}}$

Functionality:

Key Generation:

Upon receiving $(\mathsf{keygen}, \mathsf{sid}, p_i)$ from some party p_j:
Forward $(\mathsf{keygen}, \mathsf{sid}, p_i)$ to the adversary.
Upon receiving $(\mathsf{Public\ Key}, \mathsf{sid}, p_i, pk_i)$ from the adversary:
Output $(\mathsf{public\ key}, \mathsf{sid}, p_i, pk_i)$ to p_i and record the pair (p_i, pk_i).

The following rules apply only after the corresponding initial $(\mathsf{keygen}, \mathsf{sid}, p_*)$ *message*

Signature Generation:

Upon receiving $(\mathsf{sign}, \mathsf{sid}, \mathsf{ssid}, p_i, m)$ from p_i:
Forward $(\mathsf{sign}, \mathsf{sid}, \mathsf{ssid}, p_i, m)$ to the adversary.
Upon receiving $(\mathsf{signature}, \mathsf{sid}, \mathsf{ssid}, p_i, m, \sigma)$ from the adversary:
If no entry $(\mathsf{sid}, \mathsf{ssid}, m, \sigma, pk_i, 0)$ is recorded.
Record the entry $(\mathsf{sid}, \mathsf{ssid}, m, \sigma, \{pk_i\}, 1)$.
Output $(\mathsf{signature}, \mathsf{sid}, \mathsf{ssid}, p_i, m, \sigma)$ to p_i.

Signature Combination:

Upon receiving $(\mathsf{cb}, \mathsf{sid}, \mathsf{ssid}, G, m, \{(\sigma_j, pk'_j)\}_{p_j \in G})$ from some party p_i, with $G \subseteq \Psi$ and $|G| \geq 2$:
Forward $(\mathsf{cb}, \mathsf{sid}, \mathsf{ssid}, G, m, \{(\sigma_j, pk'_j)\}_{p_j \in G})$ to the adversary.
Upon receiving $(\mathsf{cb}, \mathsf{sid}, \mathsf{ssid}, G, m, \{(\phi_j)\}_{p_j \in G}, \sigma_G)$ from the adversary:
Determine $f_G \leftarrow \mathsf{BatchIndividualVerification}(\mathsf{sid}, \mathsf{ssid}, G, m, \{(\sigma_j, pk'_j, \phi_j)\}_{p_j \in G})$
If $f_G = 1$:
Record the entry $(\mathsf{sid}, \mathsf{ssid}, m, \sigma_G, \{pk_j\}_{p_j \in G}, 1)$.
Output $(\mathsf{multisignature}, \mathsf{sid}, \mathsf{ssid}, G, m, \sigma_G)$ to p_i.
Else:
Output $(\mathsf{multisignature}, \mathsf{sid}, \mathsf{ssid}, G, m, \perp)$ to p_i.

Signature Verification:

Upon receiving $(\mathsf{verify}, \mathsf{sid}, \mathsf{ssid}, G, m, \sigma_G, \{pk'_j\}_{p_j \in G})$ from some party p_j with $G \subseteq \Psi$ and $|G| \geq 1$:
Forward $(\mathsf{verify}, \mathsf{sid}, \mathsf{ssid}, G, m, \sigma_G, \{(pk'_j)\}_{p_j \in G})$ to the adversary.
Upon receiving $(\mathsf{verified}, \mathsf{sid}, \mathsf{ssid}, G, m, \{\phi_j\}_{p_j \in G}, \phi_G)$ from the adversary:
Determine $f_G \leftarrow \mathsf{Verification}(\mathsf{sid}, \mathsf{ssid}, G, m, \sigma_G, \{\phi_j\}_{p_j \in G}, \{\phi_j\}_{p_j \in G}, \phi_G)$
Output $(\mathsf{verified}, \mathsf{sid}, \mathsf{ssid}, G, m, f_G)$ to p_j.

Verification Procedure:

$\mathsf{Verification}(\mathsf{sid}, \mathsf{ssid}, G, m, \sigma, \{pk'_j\}_{p_j \in G}, \phi)$:
(1) If $\forall p_j \in G$, $pk'_j = pk_j$ and the entry $(\mathsf{sid}, \mathsf{ssid}, m, \sigma, \{pk_j\}_{p_j \in G}, 1)$ is recorded, return $f = 1$.
(Completeness: Verification succeeds for valid (multi-)signatures with the registered keys.)
(2) Else, if $\exists p_j \in G$, $pk'_j = pk_j$, and p_j is not corrupted, and no entry $(m, \sigma', \{pk'_j\}_{p_j \in G}, 1)$, for any σ', is recorded:
record the entry $(\mathsf{sid}, \mathsf{ssid}, m, \sigma, \{pk'_j\}_{p_j \in G}, 0)$ and return $f = 0$.
(Unforgeability: Verification fails if one signer did not sign m.)
(3) Else, if an entry $(\mathsf{sid}, \mathsf{ssid}, m, \sigma, \{pk'_j\}_{p_j \in G}, f')$ is recorded:
Return $f = f'$.
(Consistency: Identical parameters produce the same verification result.)
(4) Else:
Record the entry $(\mathsf{sid}, \mathsf{ssid}, m, \sigma, \{pk'_j\}_{p_j \in G}, \phi)$ and return $f = \phi$.

$\mathsf{BatchIndividualVerification}(\mathsf{sid}, \mathsf{ssid}, G, m, \{(\sigma_j, pk'_j, \phi_j)\}_{p_j \in G})$:
Determine $\{f_j \leftarrow \mathsf{Verification}(\mathsf{sid}, \mathsf{ssid}, \{p_j\}, m, \sigma_j, \{pk'_j\}, \phi_j)\}_{p_j \in G}$
Return $\bigwedge_{p_j \in G} f_j$

Functionality 6 $\mathcal{F}_{\mathrm{CA}}$: Certification Authority

Participants:

$p_1, \ldots, p_n$: Parties interacting with the functionality.
Sim: Ideal process adversary (simulator).

Local State:

Reg: A table of registered pairs (p_j, v_j), initially empty.

Events:

request $(\mathsf{Register}, \mathsf{sid}, v)$: A party registers a value v (e.g., public keys)
request $(\mathsf{Retrieve}, \mathsf{sid}, p_j)$: A party requests to retrieve the value registered by process p_j

Functionality:

Upon receiving $(\mathsf{Register}, \mathsf{sid}, v_i)$ from party p_i for the first time:
  Send $(\mathsf{Registered}, \mathsf{sid}, p_i, v_i)$ to Sim.
  Upon receiving ok from Sim:
    store (sid, p_i, v_i) in Reg.

Upon receiving $(\mathsf{Retrieve}, \mathsf{sid}, p_j)$ from party p_i:
  Send $(\mathsf{Retrieve}, \mathsf{sid}, p_i, p_j)$ to Sim.
  Upon receiving ok from Sim:
    If there exists a recorded pair (sid, p_j, v_j) in Reg:
      Send $(\mathsf{Retrieve}, \mathsf{sid}, v_j)$ to p_i.
    Else:
      Send $(\mathsf{Retrieve}, \mathsf{sid}, \perp)$ to p_i.