Round-Efficient Byzantine Agreement and Multi-Party Computation with Asynchronous Fallback

Giovanni Deligios, Martin Hirt, and Chen-Da Liu-Zhang*

{gdeligios,hirt}@inf.ethz.ch, ETH Zürich
cliuzhan@andrew.cmu.edu, Carnegie Mellon University

Abstract. Protocols for Byzantine agreement (BA) and secure multi-party computation (MPC) can be classified according to the underlying communication model. The two most commonly considered models are the synchronous one and the asynchronous one. Synchronous protocols typically lose their security guarantees as soon as the network violates the synchrony assumptions. Asynchronous protocols remain secure regardless of the network conditions, but achieve weaker security guarantees even when the network is synchronous.
Recent works by Blum, Katz and Loss [TCC'19], and Blum, Liu-Zhang and Loss [CRYPTO'20] introduced BA and MPC protocols achieving security guarantees in both settings: security up to t_s corruptions in a synchronous network, and up to t_a corruptions in an asynchronous network, under the provably optimal threshold trade-offs $t_a \leq t_s$ and $t_a + 2t_s < n$. However, current solutions incur a high synchronous round complexity when compared to state-of-the-art purely synchronous protocols. When the network is synchronous, the round complexity of BA protocols is linear in the number of parties, and the round complexity of MPC protocols also depends linearly on the depth of the circuit to evaluate.
In this work, we provide round-efficient constructions for both primitives with optimal resilience: fixed-round and expected constant-round BA protocols, and an MPC protocol whose round complexity is independent of the circuit depth.

1 Introduction

1.1 Motivation

Byzantine agreement (BA) and secure *multi-party computation* (MPC) are two fundamental and widely explored problems in distributed computing and cryptography.

The general problem of MPC allows a set of n parties to correctly carry out an arbitrary computation, without revealing anything about their inputs that could not be inferred from the computed output [45, 46]. Such guarantees must hold even when a subset of the parties are corrupted and actively deviate from the protocol specification. BA can be seen as an instance of MPC, in which the function to evaluate guarantees agreement on a common output [42, 44] and privacy is not a requirement. Protocols for BA are often used as building blocks within larger constructions, including crucially in MPC protocols, and have received renewed attention in the context of blockchain protocols (starting with [38]).

There are two prominent communication models in the literature when it comes to the design of such primitives. In the *synchronous model*, parties have synchronized clocks and messages are assumed to be delivered within some (publicly known) delay Δ. Protocols in this setting achieve very strong security guarantees: under standard setup assumptions, BA [22, 30] and MPC [4, 5, 7, 15, 18, 19, 21, 25, 26, 28, 43] are achievable even when up to $t < n/2$ parties are corrupted. However, the security of synchronous protocols is often completely compromised as soon as the synchrony assumptions are violated (for example, if even one message is delayed by more than Δ due to unpredictable network delays). This is particularly undesirable in real-world applications, where even the most stable networks, such as the Internet, occasionally experience congestion or failures. In the *asynchronous model*, no timing assumptions are needed, and messages can be arbitrarily delayed. Protocols designed in this model are robust even in unpredictable real-world networks, but the security guarantees that can be achieved are

* This work was partially carried out while the author was at ETH Zürich.

significantly weaker. For example, protocols in this realm can only tolerate up to $t < n/3$ corruptions [8, 14, 24].

As a consequence, when deploying protocols in real-world scenarios, one has to decide between employing synchronous protocols —risking catastrophic failures in the case of unforeseen network delays —or settling for the weaker security guarantees of asynchronous protocols.

1.2 Contributions

A recent line of work [9, 11] provides BA and MPC protocols that are secure up to $t_s < n/2$ corruptions when the network is synchronous, and $t_a \leq t_s$ corruptions when the network is asynchronous, for the optimal trade-off $t_a + 2t_s < n$.

Such protocols strive to achieve the best of both models, but current solutions are far from being efficient, especially when it comes to running time; in this paper, we focus on minimizing round complexity when the network is synchronous. This is of primary importance in typical scenarios, where the network is stable and synchronous most of the time, but may suffer from unexpected congestion.

Current BA and MPC protocols in this realm [9, 11] require a linear number of rounds in the number of parties. Moreover, known MPC protocols [11] also have linear round complexity in the depth of the circuit to evaluate.

This is in contrast to the efficiency of state-of-the-art purely synchronous protocols: fixed-round BA protocols (*Monte-Carlo* type) require only $O(\kappa)$ rounds, and BA protocols with probabilistic termination (*Las-Vegas* type) require an expected constant number of rounds. Furthermore, current MPC protocols only require a constant number of broadcast rounds.[1] We therefore ask the following natural question.

> *Do there exist* BA *and* MPC *protocols that are 1) round-efficient and secure for up to* $t_s < n/2$ *corruptions in a synchronous network, and 2) secure up to* $t_a < n/3$ *corruptions in an asynchronous network?*

We answer this question affirmatively by providing the following results.

Round-Efficient Synchronous BA with Asynchronous Fallback. We obtain the first BA protocols in this realm that are round efficient when the network is synchronous and with the optimal trade-off $t_a + 2t_s < n$, by providing fixed-round and expected constant-round constructions. In doing so, we completely characterize the feasibility of a primitive that we believe to be of independent interest: a round-efficient BA that is secure in a synchronous network for up to t_s-corruptions, and retains some weak validity guarantee even in an asynchronous network up to t_a-corruptions. We show that its optimal tradeoff is $2t_a + t_s < n$ and $t_s < n/2$. As a side result, we also provide a simpler construction of the primitive for the trade-off $t_a + 2t_s < n$. We then use this primitive as a fundamental building block to design further round-efficient primitives: broadcast protocols with similar guarantees, and also synchronous BA/MPC protocols with asynchronous fallback.

Round-Efficient Synchronous MPC with Asynchronous Fallback. We obtain the first synchronous MPC protocol with asynchronous fallback with optimal guarantees (i.e. $t_a+2t_s < n$ and $(n-t_s)$-output quality as in [11]) that requires a constant number of all-to-all broadcast/BA invocations. In particular, the round complexity is independent of the depth of the circuit. When instantiating the broadcast/BA protocols with our constructions (in their fixed-round version), we achieve a total round complexity of $O(\kappa)$.[2] For this, we adapt techniques based on garbled circuits [46, 5, 20] to our setting.

[1] This is when requiring full security. When striving for weaker security guarantees, such as security with abort, there are solutions that run in a constant number of rounds (e.g. [3]).

[2] Achieving such MPC constructions in the expected constant-round realm requires composing protocols with probabilistic termination in a round-preserving fashion. We leave this interesting line of research for future work. See [32, 16] for interesting discussions and challenges in this setting.

1.3 Related Work

Protocols achieving security guarantees in both synchronous and asynchronous networks have only begun to be studied in relatively recent works. Closest to ours are works by Blum et al. [9, 11], which consider the problem of BA and MPC achieving security guarantees in both communication models. Our work improves upon the round efficiency of these protocols. In the same setting, the work in [10] considers the problem of state-machine replication (SMR).

The work in [27] introduces a variant of the purely synchronous model, which allows for network partitions, motivated by eclipse attacks. In this model, the adversary is allowed to disconnect a certain fraction of parties from the rest of the network in each round. BA and MPC protocols tolerating the optimal corruption threshold in this model are also provided. In [2], similar results are achieved for SMR. These results are crucially different from ours, as they rely on the fact that synchrony is maintained in part of the network. In contrast, our protocols give guarantees even if the network is fully asynchronous.

Other works that provide hybrid security guarantees include BA achieving guarantees in synchronous or partially synchronous networks [36], or guarantees against active corruptions in a synchronous network and fail-stop in an asynchronous network [33].

A different line of work [39, 40, 35, 34] has recently investigated protocols that achieve *responsiveness*. These protocols operate under a synchronous network, and provide the additional guarantee that parties obtain output as fast as the network delay allows. Note that these works do not provide any security guarantees when the network is not synchronous.

2 Model

We consider a set of n parties $\mathcal{P} = \{P_1, \dots, P_n\}$. We denote by κ the security parameter.

2.1 Communication and Adversarial Models

We consider a complete network of authenticated channels. Our protocols strive to be secure in the two main communication models in the literature: the *synchronous* and the *asynchronous* models.

In the synchronous model, parties have access to synchronized clocks, and all messages are delivered within a known delay $0 \leq \Delta \in \mathbb{R}$. In this setting, protocols can be conveniently described as proceeding in *rounds*: parties begin the protocol simultaneously, and the r-th round identifies the time interval $[(r-1)\Delta, r\Delta)$ for all integers $r \geq 1$. If a party receives a message within this time interval, we say they *receive a message in round r*. When a party *sends a message in round r*, it means they send it at time $(r-1)\Delta$. Within each round, the adversary can schedule the delivery of messages arbitrarily. In particular, we consider a *rushing* adversary that generates the messages of corrupted parties after seeing all messages sent by honest parties.

In the asynchronous model, parties do not have access to synchronized clocks, and the adversary can schedule the delivery of messages arbitrarily. However, the adversary cannot drop messages, meaning that all messages are eventually delivered.

We consider a *static* adversary that can corrupt parties in an arbitrary manner at the beginning of the protocol.[3]

2.2 Cryptographic Primitives

Public-Key Infrastructure. We assume that parties have access to a public-key infrastructure. This means parties agree on a set of public keys $(\mathtt{pk}_1, \dots, \mathtt{pk}_n)$ and party P_i holds the secret key $\mathtt{sk}_i$ associated with $\mathtt{pk}_i$.

[3] However, note that our protocols for BA are adaptively secure.

Definition 1. *A (digital) signature scheme is a triple of algorithms* $(\mathsf{Sgn}, \mathsf{Vfy}, \mathsf{Kgn})$ *such that:*

- *given the security parameter* κ*, the key generation algorithm* Kgn *outputs a public/secret key pair* $(\mathtt{pk}, \mathtt{sk}) \in \mathcal{PK} \times \mathcal{SK}$*;*
- *given a secret key* $\mathtt{sk} \in \mathcal{SK}$ *and a message* $m \in \{0,1\}^*$*, the signing algorithm* Sgn *outputs* $\mathcal{S} \ni \sigma := \mathsf{Sgn}(m, \mathtt{sk})$*;*
- *given a message* $m \in \{0,1\}^*$*, a public key* $\mathtt{pk} \in \mathcal{PK}$*, and a signature* $\sigma \in \mathcal{S}$*, the verifying algorithm* Vfy *outputs* $\mathsf{Vfy}(m, \sigma, \mathtt{pk}) \in \{0,1\}$*;*
- $\mathsf{Vfy}(m, \sigma, \mathtt{pk}) = 1$ *if and only if* $\sigma = \mathsf{Sgn}(m, \mathtt{sk})$ *where* $(\mathtt{pk}, \mathtt{sk})$ *is a key pair output by* Kgn*.*

We require the signature scheme to be unforgeable against chosen message attacks.

Coin-Flip. Parties have access to a Coin-Flip functionality, parametrized by t, that allow mutually distrustful parties to generate a common uniformly random bit.

Functionality $\mathcal{F}^t_{\mathsf{CoinFlip}}$

Let k be a non-negative integer. Upon receiving message k from at least $t+1$ distinct parties, sample coin_k uniformly at random from $\{0,1\}$ and send message (k, coin_k) to all parties.

Such a functionality can be realized in the asynchronous model (e.g. [14, 1, 41]) under general assumptions up to $t < n/3$ corruptions, or even 1-round using unique threshold signatures in the random oracle model (e.g. [12]) up to $t < n/2$ corruptions.

3 Definitions

The definitions we give are somewhat non-standard, out of necessity to allow for different abort behaviors depending on the network condition (synchronous/asynchronous), which is unknown to the parties at the start of the protocol. If an honest party outputs symbol $\top$, this means they detected (during the execution) that the network is asynchronous. Whenever desirable, our definitions are equivalent to the standard notions.

3.1 Agreement Primitives

Byzantine agreement (BA) allows a set of parties (each holding an input) to agree on a common value, even when a subset of parties has arbitrary behavior.

Definition 2. *(Byzantine agreement) Let* Π *be a protocol executed by parties* $P_1, \ldots, P_n$ *where each party* P_j *holds input* $v_j \in \{0,1\}$ *and terminates upon generating an output* $f_j \in \{0,1,\top\}$*. We say protocol* Π *achieves*

- *(t-**validity**) if whenever up to t parties are corrupted: if there is v such that each honest party holds input $v_j = v$, then every honest party outputs $f_j = v$.*
- *(t-**weak validity**) if whenever up to t parties are corrupted: if there is v such that each honest party holds input $v_j = v$, then every honest party outputs $f_j \in \{v, \top\}$.*
- *(t-**consistency**) if whenever up to t parties are corrupted: there is $v \in \{0,1,\top\}$ such that each honest party outputs $f_j = v$.*
- *(t-**liveness**) if whenever up to t parties are corrupted: no honest party outputs $f_j = \top$.*

Together, the t-consistency and t-liveness properties imply the more widely adopted consistency notion. If a protocol Π achieves t-validity, t-consistency, and t-liveness, we say it achieves t-*security* (or that it is t-*secure*).

Weak consensus (WC) is a primitive that achieves a weaker form of agreement compared to BA: it guarantees agreement among all the parties that output a bit, but parties are allowed to output a special symbol $\bot$.

Definition 3. *(Weak consensus) Let Π be a protocol executed by parties $P_1, \ldots, P_n$ where each party P_j holds input $v_j \in \{0,1\}$ and terminates upon generating an output $f_j \in \{0,1,\bot,\top\}$. We say protocol Π achieves*

- *(t-**validity**) if whenever up to t parties are corrupted: if there is v such that each honest party holds input $v_j = v$, then each honest party outputs $f_j = v$.*
- *(t-**weak validity**) if whenever up to t parties are corrupted: if there is v such that each honest party holds input $v_j = v$, then all honest parties output $f_j \in \{v, \top\}$.*
- *(t-**weak consistency**) if whenever up to t parties are corrupted: if an honest party outputs $f_j = v \in \{0,1\}$, no honest party outputs $f_j = 1 - v$.*
- *(t-**liveness**) if whenever up to t parties are corrupted: no honest party outputs $f_j = \top$.*

3.2 Broadcast Primitives

Broadcast (BC, sometimes called *Byzantine broadcast*) allows a designated party, called the *sender*, to consistently send a message to multiple parties in the presence of active adversarial behavior.

Definition 4. *(Broadcast) Let Π be a protocol executed by parties $P_1, \ldots, P_n$ where a designated party P^* holds input $v^* \in \{0,1\}$ and each party P_j terminates upon generating an output $f_j \in \{0,1,\top\}$. We say protocol Π achieves*

- *(t-**validity**) if whenever up to t parties are corrupted: if the sender P^* is honest, then each honest party outputs $f_j = v^*$.*
- *(t-**weak-validity**) if whenever up to t parties are corrupted: if the sender P^* is honest, then each honest party outputs $f_j \in \{v^*, \top\}$.*
- *(t-**consistency**) if whenever up to t parties are corrupted: there is $v \in \{0,1,\top\}$ such that each honest party outputs $f_j = v$.*

Gradecast (GBC) is a primitive that is similar to broadcast, but achieves a weaker form of consistency guarantees.

Definition 5. *(Gradecast) Let Π be a protocol executed by parties $P_1, \ldots, P_n$ where a designated sender P^* holds input $v^* \in \{0,1\}$ and each party P_j terminates upon generating an output value and a grade $(f_j, g_j) \in \{0,1,\bot\} \times \{0,1,2\}$. We say protocol Π achieves*

- *(t-**graded validity**) if whenever up to t parties are corrupted: if P^* is honest, then all honest parties output $(v^*, 2)$.*
- *(t-**graded consistency**) if whenever up to t parties are corrupted:*
  a. *there is a $v \in \{0,1\}$ such that all honest parties output either $(v,2)$, $(v,1)$ or $(\bot,0)$.*
  b. *if some honest party outputs $(v,2)$ for any $v \in \{0,1\}$, no honest party outputs $(\bot,0)$.*
- *(t-**weak-graded validity**) if whenever up to t parties are corrupted in an execution of Π: if P^* is honest, then each honest party outputs either $(v^*,2)$, $(v^*,1)$ or $(\bot,0)$.*

3.3 Multi-party Computation

A protocol for multi-party computation (MPC) allows a set of n mutually distrustful parties (each holding an input v_i) to correctly compute a function $g(v_1, \ldots, v_n)$ without revealing anything about their inputs that could not be inferred from the output. The security of MPC is usually described in the UC framework [13]. At a high-level, a protocol is secure if it is "indistinguishable" from an ideal functionality with the desired properties.

We recall the ideal functionality for MPC with *full security* (where parties are guaranteed to obtain the correct output), and with L-output quality (the number of inputs taken into account for the computation), as introduced in [11].

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

Let $\mathcal{P}$ be the set of parties and let $f : (\{0,1\}^* \cup \{\bot\})^n \to \{0,1\}^*$ be the function to be evaluated. For each $P_i \in \mathcal{P}$ set $x_i = y_i := \bot$. Set $S := \mathcal{P}$.

```
1: On input $(\texttt{input}, v)$ from party $P_i \in \mathcal{P}$, set $x_i := v$ and output $(\texttt{input}, P_i)$ to the adversary.
2: On input $(\texttt{output-set}, S')$ from the (ideal) adversary, where $S' \subseteq \mathcal{P}$, and $\#S' = L$, set $S := S'$ and
   $x_i := \bot$ for all $P_i \notin S'$.
3: Once all honest parties in $S$ have provided input, set each $y_i = f(x_1, \ldots, x_n)$.
4: On input $(\texttt{get-output})$ from party $P_i$, output $(\texttt{output}, y_i, \text{sid})$ to party $P_i$.
```

A weaker notion of security is also of interest. In MPC *with unanimous output*, the ideal world adversary can choose whether all honest parties receive the correct output or they all receive symbol $\top$. We denote ideal functionality describing this security notion by $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{uout},L}$.

Definition 6. *An* MPC *protocol Π achieves t-full security (respectively t-unanimous output) with L-output quality if it* UC*-realizes functionality $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{sec},L}$ (respectively $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{uout},L}$), whenever up to t parties are corrupted in an execution of Π.*

4 Round-Efficient Byzantine Agreement with Asynchronous Weak Validity

We study the feasibility and efficiency of BA protocols that are t_s-secure when the network is synchronous, and at the same time achieve t_a-weak validity when the network is asynchronous. This primitive is of independent interest, as it is used to construct BA protocols with asynchronous fallback (see Section 5). Moreover, it turns out to be fundamental in the design of further distributed protocols, for example to obtain constant-round synchronous broadcast protocols with asynchronous weak validity (see Section G), which in turn are used to construct synchronous MPC protocols with asynchronous fallback [11].

In this section, we completely characterize the threshold conditions under which such a primitive exists, and provide different round-efficient constructions (fixed-round and with probabilistic termination).

First, in Section 4.2, we show a fixed-round BA protocol that runs in $O(\kappa)$ rounds when the network is synchronous. Then, in Section F, we show a version running in expected constant-rounds when the network is synchronous.[4]

While the optimal achievable trade-off (see [9]) of a BA protocol with full asynchronous fallback security is $t_a + 2t_s < n$ and $t_a \le t_s$ (which together imply $t_s < n/2$), we show that there is room for improvement when only requiring asynchronous weak-validity. In this case, we prove the optimal threshold trade-off to be $2t_a + t_s < n$ and $t_s < n/2$.

4.1 Weak Consensus with Asynchronous Weak Validity

The main tool in our BA constructions is a round-based weak consensus protocol that is secure in a synchronous network (up to t_s-corruptions), and achieves weak validity even if the network is asynchronous (up to t_a-corruptions).

In traditional weak-consensus, parties are allowed to output a symbol $\bot$, signaling they are unsure about what bit to output. We also allow parties to output symbol $\top$, which also signals a lack of information necessary to reach agreement, but only due to the network being asynchronous. Distinguishing between these two outcomes is essential, but not trivial. The reason is that, when designing round-based protocols, if the network is asynchronous one cannot take advantage of eventual delivery of messages, since parties only wait for a fixed amount of time Δ per round. Therefore, when an expected message is not delivered within a round, parties cannot decide if 1) the network is synchronous and the sender is corrupted, or 2) the network is asynchronous and the message was delayed by the adversary.

[4] When the network is asynchronous, the adversary can delay messages for any arbitrary (but finite) amount of time, and so the protocols may run for longer.

We address this problem by making use of a gradecast (GBC) protocol that achieves graded validity and graded consistency when the network is synchronous, and weak-graded validity when the network is asynchronous (see Section B).

By requiring each party to gradecast their input, we can have parties take a non-$\top$ decision only if they receive at least $n - t_s$ values with grade 2. Indeed, if the network is synchronous, honest parties output grade 2 in all executions with honest senders. Therefore, less than $n - t_s$ outputs with grade 2 guarantee that the network is asynchronous and it is safe to output $\top$.

In case at least $n - t_s$ values with grade 2 are received, the output determination ensures the required guarantees: party P_i outputs v if 1) they received $(v, 2)$ from $n - t_s$ gradecasts, or 2) they received $(v, 2)$ from at least $n - t_s - t_a$ gradecasts and $(1 - v, \cdot)$ from up to t_a; in any other case they output $\perp$.

In particular, if the network is asynchronous and there are up to t_a corruptions, weak validity is achieved: any party that does not output $\top$ has received at least $n - t_s$ values with grade 2, and $n - t_s - t_a > t_a$ of those values correspond to the inputs of honest parties. Moreover, when the network is synchronous and up to t_s parties are corrupted, there cannot be honest parties P_i and P_j that output different bits v and $1 - v$, respectively. This is because 1) if P_i receives $(1 - v, \cdot)$ up to t_a times, then P_j cannot receive $(1 - v, 2)$ more than t_a times, and 2) if P_i receives $(v, 2)$ at least $n - t_s$ times, then P_j receives $(v, \cdot)$ at least $n - t_s > t_a$ times.

We formally describe the protocol below. Let Π^t_{GBC} be a gradecast protocol running in s rounds. The n executions of Π^t_{GBC} are to be run in parallel to preserve round-efficiency. Security is proven in Section C.

Protocol $\Pi^{t_a,t_s}_{\mathsf{WC}}\left(\Pi^{\max\{t_a,t_s\}}_{\mathsf{GBC}}\right)$

We describe the protocol from the point of view of party P_j holding input v_j. We denote by $\Pi^{\max\{t_a,t_s\}}_{\mathsf{GBC}}(j)$ an execution of protocol $\Pi^{\max\{t_a,t_s\}}_{\mathsf{GBC}}$ in which party P_j acts as the sender.

Inizialization step. Set $b_j := \top$. For $b \in \{0,1\}$ set $S^b_j := \varnothing$, $U^b_j := \varnothing$.

Rounds 1 to s.

```
1: for $1 \le i \le n$ do
2:     $w_{ij} := (b_{ij}, g_{ij}) := \Pi^{\max\{t_a,t_s\}}_{\mathsf{GBC}}(i)$;
3:     if $w_{ij} = (0,2)$ then $S^0_j := S^0_j \cup \{b_{ij}\}$;
4:     end if
5:     if $w_{ij} = (1,2)$ then $S^1_j := S^1_j \cup \{b_{ij}\}$;
6:     end if
7:     if $w_{ij} = (0,1)$ then $U^0_j := U^0_j \cup \{b_{ij}\}$;
8:     end if
9:     if $w_{ij} = (1,1)$ then $U^1_j := U^1_j \cup \{b_{ij}\}$;
10:    end if
11: end for
```

Output determination.

```
1: if $\#(S^0_j \sqcup S^1_j) \ge n - t_s$ then
2:     if there is $b \in \{0,1\}$ such that $\#S^b_j \ge n - t_s$ then
3:         $b_j; = b$;
4:     else if there is $b \in \{0,1\}$ such that $\#S^b_j \ge n - t_s - t_a$ and $\#(S^{1-b}_j \sqcup U^{1-v}_j) \le t_a$ then
5:         $b_j := b$;
6:     else $b_j := \perp$;
7:     end if
8: end if
9: output $b_j$ and terminate;
```

Lemma 1. *Assume protocol* $\Pi^{\max\{t_a,t_s\}}_{\mathsf{GBC}}$ *achieves the following security guarantees.*

– *When run over a synchronous network:* $(\max\{t_s, t_a\})$*-graded validity and* $(\max\{t_s, t_a\})$*-graded consistency.*

– *When run over an asynchronous network:* $(\max\{t_s, t_a\})$*-weak graded validity.*

Then, if $2t_a + t_s < n$ *and* $t_s < n/2$*, protocol* $\Pi_{\mathsf{WC}}^{t_a,t_s}\left(\Pi_{\mathsf{GBC}}^{\max\{t_a,t_s\}}\right)$ *achieves the following security guarantees.*

– *When run over a synchronous network:* t_s*-liveness,* t_s*-validity,* t_s*-weak consistency.*
– *When run over an asynchronous network:* t_a*-weak validity.*

When assuming a worse tradeoff $t_a + 2t_s < n$ and $t_s \le t_a$ (which is optimal to achieve BA with full asynchronous fallback) one can obtain a simpler and more efficient weak consensus protocol with asynchronous weak validity (see Section D for a construction and security proof).

4.2 Fixed-Round Synchronous BA with Asynchronous Weak Validity

We now present a fixed-round synchronous Byzantine agreement protocol with asynchronous weak validity. If the network is synchronous and there are up to t_s corruptions, agreement is reached with overwhelming probability after $O(\kappa)$ rounds. Moreover, even when the network is asynchronous and there are up to t_a corruptions, the protocol achieves weak validity.

Following the traditional Feldman-Micali paradigm [23], parties run a sequence of iterations. Each iteration consists of a weak consensus protocol $\Pi_{\mathsf{WC}}^{t_a,t_s}$ followed by an invocation to the coin-flip functionality $\mathcal{F}_{\mathsf{CoinFlip}}^{t_s}$, where: 1) parties that obtain a bit as output of $\Pi_{\mathsf{WC}}^{t_a,t_s}$ keep this value for the next iteration, 2) parties that obtained $\bot$ adopt the value of the coin, and 3) parties that obtained $\top$ keep their initial value of the iteration.

Notice that, if the network is synchronous, the output of an honest party in the execution of $\Pi_{\mathsf{WC}}^{t_a,t_s}$ is binary or $\bot$. Since weak consensus guarantees that honest parties do not output contradicting bits, and the coin value is uniform and independent of the output of weak consensus, agreement is reached with probability 1/2 per iteration.[5]

Moreover, if the network is asynchronous, weak validity is achieved. The reason is that in each iteration, if all honest parties start with the same value v, then weak validity of $\Pi_{\mathsf{WC}}^{t_a,t_s}$ ensures that they all output v or $\top$, and the coin value is ignored. Therefore, they keep v as the value for the next iteration.

We formally describe the protocol below. Security is proven in Section E.

Protocol $\Pi_{\mathsf{SBA}}^{t_a,t_s}\left(\Pi_{\mathsf{WC}}^{t_a,t_s}, \mathcal{F}_{\mathsf{CoinFlip}}^{t_s}\right)$

We describe the protocol from the point-of-view of party P_j holding input v_j.

Initialization step: Set $b_j := v_j$.

```
1: for $k = 1$ to $\kappa$ do
2:     $b_j := \Pi_{\mathsf{WC}}^{t_a,t_s}(b_j)$
3:     $(k, \mathsf{coin}_k) := \mathcal{F}_{\mathsf{CoinFlip}}^{t_s}(k)$
4:     if $b_j = \bot$ then
5:         $b_j := \mathsf{coin}_k$
6:     else if $b_j = \top$ then
7:         $b_j := v_j$
8:     end if
9: end for
10: Output $b_j$
```

Lemma 2. *Assume protocol* $\Pi_{\mathsf{WC}}^{t_a,t_s}$ *achieves the following security guarantees.*

– *When run over a synchronous network:* t_s*-liveness,* t_s*-validity, and* t_s*-weak consistency.*

[5] For simplicity, we describe our protocols and proofs assuming an ideal coin flip that outputs a common uniform random bit to all honest parties in one round (e.g. [12]). If a q-weak coin flip is used instead, where honest parties agree with probability q, the round complexity increases by a factor of $O(1/q)$.

– *When run over an asynchronous network: t_a-weak validity.*

Then, protocol $\Pi_{\mathsf{SBA}}^{t_a,t_s}\left(\Pi_{\mathsf{WC}}^{t_a,t_s}, \mathcal{F}_{\mathsf{CoinFlip}}^{t_s}\right)$ *achieves the following security guarantees with overwhelming probability.*

– *When run over a synchronous network: t_s-security. Moreover, the protocol runs in $O(\kappa)$ rounds and achieves simultaneous termination.*
– *When run over an asynchronous network: t_a-weak validity.*

4.3 Optimality of Synchronous BA with Asynchronous Weak Validity

In this section we prove the optimality of our constructions with respect to corruption thresholds. More specifically, we show that the tradeoff assumption $2t_a + t_s < n$ is not only sufficient, but also necessary to obtain BA protocols that are secure up to t_s corruptions in a synchronous network, and achieve weak validity up to t_a corruptions in an asynchronous network.

Lemma 3. *Assume $2t_a + t_s \geq n$. There does not exist an n-party Byzantine agreement protocol that is both*

– *t_s-secure when run over a synchronous network;*
– *t_a-weakly valid when run over a synchronous network*

Proof. Assume there exists a protocol Π achieving all the above security guarantees. Partition the party set $\mathcal{P}$ into sets S_a, S_b and K where $\#S_a = \#S_b = t_a$ and $\#K = t_s$.

– **Scenario** 1. The network is synchronous. Parties in S_a participate in Π with input 0. Parties in S_b participate in Π with input 1. Parties in K are corrupted by the adversary and simply abort.
– **Scenario** 2. All messages from parties in K are dropped (delayed for longer than the round time) by the adversary. Parties in S_a participate in Π with input 0. Parties in S_b are corrupted by the adversary, but participate in Π as if they were honest with input 1. Parties in K partecipate in the protocol with input 0.
– **Scenario** 3. All messages from parties in K are dropped (delayed for time $\delta > \Delta$) by the adversary. Parties in S_b participate in Π with input 1. Parties in S_a are corrupted by the adversary and participate in Π as if they were honest with input 0. Parties in K participate in the protocol using input 1.

In scenario 1, parties in S_a and S_b output the same value $b_1 \in \{0, 1\}$ by t_s-consistency and t_s-liveness of Π. In scenario 2, parties in S_a output 0 or $\top$ by t_a-weak validity of Π. In scenario 3, parties in S_b output 1 or $\top$ by t_a-weak validity of Π. Since the views of parties in S_a in scenarios 1 and 2 are indistinguishable, and the views of parties in S_b in scenarios 1 and 3 are indistinguishable, then in scenario 2 (respectively 3) no party in S_a (respectively S_b) outputs $\top$. However, this means that $0 = b_1 = 1$, which is a contradiction (here, we assumed parties are deterministic, but the same argument can be adapted to probabilistic parties and their output distributions). □

5 Synchronous BA with Asynchronous Fallback

In order to achieve a BA protocol that is t_s-secure when the network is synchronous, and t_a-secure when the network is asynchronous, we use the compiler $\Pi_{\mathsf{HBA}}^{t_a,t_s}$ introduced by Blum et al. [9]. The compiler assumes 1) a t_s-secure synchronous BA protocol Π_1 that is t_a-weakly valid even when run over an asynchronous network, and 2) a t_a-secure asynchronous BA protocol Π_2 that achieves validity and terminates for a higher corruption threshold $t_s \geq t_a$ when the network is synchronous. The idea is to run, in sequence, the synchronous BA protocol followed by the asynchronous one. The output from the first protocol is used as input to the second.

Intuitively, when the network is synchronous, security is provided by the synchronous protocol, and preserved by t_s-validity with termination of the asynchronous one. On the other hand, when the network experiences delays, security is provided by the asynchronous protocol, while t_a-weak validity of the round-based protocol ensures an adversary cannot break pre-agreement among honest parties.

Protocol $\Pi_{\mathsf{HBA}}^{t_a,t_s}(\Pi_1, \Pi_2)$

We describe the protocol from the point of view of party P_j holding input v_j.

```
1: $b_j := \Pi_1(v_j)$;
2: if $b_j = \top$ then
3:     $b_j := v_j$;
4: end if
5: $b_j := \Pi_2(b_j)$;
6: output $b_j$;
```

Lemma 4 ([9], Theorem 3). *Assume protocol Π_1 achieves the following security guarantees.*

- *When run over a synchronous network: t_s-security.*
- *When run over an asynchronous network: t_a-weak validity.*

Furthermore, assume protocol Π_2 achieves the following security guarantees.

- *When run over a synchronous network: t_s-validity with termination.*
- *When run over an asynchronous network: t_a-security.*

Then, if $t_a \leq t_s$ and $t_a + 2t_s < n$, protocol $\Pi_{\mathsf{HBA}}^{t_a,t_s}(\Pi_1, \Pi_2)$ achieves the following security guarantees.

- *When run over a synchronous network: t_s-security.*
- *When run over an asynchronous network: t_a-security.*

By using our round-efficient synchronous BA protocols as the Π_1 component of the compiler (the fixed-round version in Section 4.2, or the expected constant-round version in Section F), and the asynchronous protocols with increased validity from [9] as the second component Π_2,[6] we obtain the following corollaries.

Corollary 1. *Let $t_a \leq t_s$ and $t_a + 2t_s < n$. There exists a protocol that achieves the following security guarantees with overwhelming probability.*

- *When run over a synchronous network: t_s-security. Moreover, it runs in $O(\kappa)$ rounds and achieves simultaneous termination.*
- *When run over an asynchronous network: t_a-security.*

Corollary 2. *Let $t_a \leq t_s$ and $t_a + 2t_s < n$. There exists a protocol that achieves the following security guarantees with overwhelming probability.*

- *When run over a synchronous network: t_s-security. Moreover, it runs in expected constant number of rounds.*
- *When run over an asynchronous network: t_a-security.*

[6] The asynchronous protocol described there has probabilistic termination and runs in an expected constant number of rounds when the network is synchronous. It is straightforward to achieve a variant of the protocol that runs in $O(\kappa)$ rounds when the network is synchronous, following Section 4.2, by substituting the weak consensus protocol with the increased-validity graded consensus protocol from [9].

6 Round-Efficient MPC with Asynchronous Fallback

Blum et al. [11] obtain the first MPC protocol that is t_s-secure in a synchronous network, and t_a-secure with $(n-t_s)$-output quality in an asynchronous network (these guarantees are provably optimal).

Their protocol requires black-box access to 1) a Byzantine agreement primitive that is t_s-secure in a synchronous network and t_a-secure in an asynchronous network, and 2) a broadcast primitive that is t_s-secure in a synchronous network and t_a-weakly valid in an asynchronous network. Their constructions for these primitives (borrowed from [9]) both require $O(n)$ rounds. Moreover, their protocol evaluates the circuit in a gate-by-gate fashion, and therefore requires $O(d)$ communication rounds, where d denotes the multiplicative depth of the circuit representing the function to evaluate.

Using our fixed-round BA and broadcast from Section 4.2 and Section G, and adapting Yao's garbled circuit techniques [46], we obtain the first MPC protocol in this realm with optimal security guarantees and that has a total round complexity of $O(\kappa)$, independent of the circuit depth. We loosely follow the structure of [17].

6.1 Multi-Party Garbled Circuits

Let g denote the function to evaluate, represented as a boolean circuit circ_g containing only `NAND` gates.[7] In general, the circuit-depth d of circ_g depends on g, and MPC protocols following the gate-by-gate paradigm typically require $O(d)$ communication rounds. Using garbled circuit techniques,[8] we obtain an MPC protocol with round complexity independent of d.

The high-level idea is to use MPC to evaluate a function f_{GRBL} that produces a garbled version of circ_g, which parties can then evaluate locally. As we will discuss, the function f_{GRBL} can be represented as a circuit whose depth is independent of g.

Roughly speaking, f_{GRBL} outputs an encrypted version of circ_g, in which all entries of each function table are encrypted using secret keys associated with the corresponding input values. A party holding two input values to a gate, together with the corresponding keys, can decrypt the corresponding entry of the function table and obtain the output of the gate (and the corresponding key). To preserve privacy, the values travelling on each wire are *masked* by `XOR`ing them with random bits. If a party is entitled to learn an output, they are given the corresponding random mask.

The function f_{GRBL} can be represented by a constant-depth circuit. The reason is that once the secret masks and keys for each wire have been generated, the garbled function tables of circ_g can be computed in parallel.

Distributed Encryption. There is a complication with the approach we described. To compute encryptions of the function table entries within the MPC, parties need white-box access to an encryption scheme. This is undesirable in itself, but matters are worsened by the fact that the circuit-representations of even the most efficient block-ciphers are fairly large ($\sim$ 6400 `AND` gates for AES-128),[9] making this approach unfeasible.

To overcome this problem, we use a distributed encryption technique due to Damgård and Ishai [20]. Let m denote a plaintext. Instead of computing $\mathsf{Enc}_k(m)$ within the multi-party computation, m is shared among the parties by means of a secret-sharing scheme (see Section A, Definition 8). Party P_i receives a share $[m]_i$ and a secret key K_i as output of the computation, and locally computes $c_i = \mathsf{Enc}_{K_i}([m_i])$. Each party then sends their encrypted shares to all parties. Upon receiving a sufficient number of encrypted shares, a party in possession of the

[7] This is without loss of generality, since any arithmetic circuit can be transformed into a boolean one, and the set {`NAND`} is functionally complete.

[8] Yao first introduced *garbled circuits* in talks related to his paper [46], but they do not explicitly appear in the paper. For a formal treatment, cf. [6].

[9] Personal communication with Yehuda Lindell.

necessary keys can decrypt them and reconstruct the secret (for example, if P_j is entitled to know the secret, they receive the keys as output from the multi-party computation). This approach extends to the dual-key setting (see Definition 7, Section A), and only requires black-box access to the encryption scheme.

Information Checking Protocol. Moving encryption outside the MPC comes at the price of secret-sharing the plaintexts to preserve privacy. In our setting, the secrets are the entries of each function table of circ_g, together with the key associated with the output value. Since we work (at least when the network is synchronous) with an honest majority, authentication of the shares is necessary to prevent corrupted parties to tamper with the reconstruction phase. This can be achieved by requiring the dealer (in our setting, f_{GRBL}) to sign the shares using digital signatures, but computing signatures within the MPC of f_{GRBL} is also inefficient.

Instead, one can use the *Information Checking Protocol* by Rabin and Ben-Or [43]. It works over a finite field $\mathbb{F}_q$. For a secret s, the dealer samples uniformly random elements $(\mathfrak{b}, \mathfrak{y})$, and computes $\mathfrak{c} = s + \mathfrak{b}\mathfrak{y}$. Party P_i is given $(s, \mathfrak{y})$: the *authentication vector*. Another party P_j (to whom P_i wishes to forward s at a later time), is given $(\mathfrak{b}, \mathfrak{c})$: the *check vector*. Upon receiving the couple $(s, \mathfrak{y})$ from P_i, party P_j can check that $\mathfrak{c} = s + \mathfrak{b}\mathfrak{y}$. If P_i is corrupted and wants to send $s' \neq s$ to P_j, party P_i has to guess the unique $\mathfrak{y}'$ solving $\mathfrak{c} = s'\mathfrak{b} + \mathfrak{y}'$, which they can only do with probability $1/(q-1)$, as the field element $\mathfrak{c} - s'\mathfrak{b}$ is distributed according to $\mathfrak{b}$.

The resulting function f_{GRBL} is formally described below. The wires of circ_g are denoted by lower-case greek letters $(\alpha, \beta, \gamma, \dots)$, and the gates with lower case english letters $(a, b, c, \dots)$. The input b_i of party P_i is a vector containing a boolean encoding of their input to g as well as extra inputs needed to generate randomness.

Function $f_{\mathsf{GRBL}}(\mathsf{circ}_g; b_1, \dots, b_n)$

Input. For each input wire ω of circ_g, let b_ω denote the corresponding input bit.

Random values. For each wire γ of circ_g generate 2 vectors of n random sub-keys $K_\gamma^0 := \left(K_\gamma^{0,1}, \dots, K_\gamma^{0,n}\right)$, $K_\gamma^1 := \left(K_\gamma^{1,1}, \dots, K_\gamma^{1,n}\right)$ and a uniform random mask $m_\gamma \in \{0,1\}$. For each gate a, for all couples (P_i, P_j) of parties, and for all $(x,y) \in \{0,1\}^2$, generate uniformly random $\mathbb{F}_q$ elements $\mathfrak{b}_a^{xy,ij}$, $\mathfrak{y}_a^{xy,ij}$.
Set $\mathfrak{B}_a^{ij} := \left(\mathfrak{b}_a^{00,ij}, \mathfrak{b}_a^{01,ij}, \mathfrak{b}_a^{10,ij}, \mathfrak{b}_a^{11,ij}\right)$ and $\mathfrak{Y}_a^{ij} := \left(\mathfrak{y}_a^{00,ij}, \mathfrak{y}_a^{01,ij}, \mathfrak{y}_a^{10,ij}, \mathfrak{y}_a^{11,ij}\right)$.

Input wires. For each input wire ω of circ_g compute $z_\omega := b_\omega \oplus m_\omega$.

Garbled function tables. For each gate a with input wires α, β and output wire γ do:

1. for all $(x,y) \in \{0,1\}^2$ compute $z_\gamma^{xy} := ((x \oplus m_\alpha)\mathtt{NAND}(y \oplus m_\beta)) \oplus m_\gamma$;
2. set $t_a^{xy} := \left(z_\gamma^{xy}, K_\gamma^{z_\gamma^{xy}}\right)$ and $T_a := \left\{t_a^{00}, t_a^{01}, t_a^{10}, t_a^{11}\right\}$;
3. compute a (t_s+1)-sharing of T_a (i.e. of each entry). Let $[t_a^{xy}]_i$ denote the i-th shares, and let $[T_a]_i$ denote the vector $\left([t_a^{00}]_i, [t_a^{01}]_i, [t_a^{10}]_i, [t_a^{11}]_i\right)$;
4. compute $\mathfrak{c}_a^{xy,ij} := [t_a^{xy}]_i + \mathfrak{b}_a^{xy,ij}\mathfrak{y}_a^{xy,ij}$. Set $\mathfrak{C}_a^{ij} := \left(\mathfrak{c}_a^{00,ij}, \mathfrak{c}_a^{01,ij}, \mathfrak{c}_a^{10,ij}, \mathfrak{c}_a^{11,ij}\right)$.

Public Outputs. For each input wire ω the masked input z_ω and the key $K_\omega^{z_\omega} = \left(K_\omega^{z_\omega,1}, \dots, K_\omega^{z_\omega,n}\right)$.

Private Outputs. For each wire γ, party P_j receives sub-keys $\left(K_\gamma^{0,j}, K_\gamma^{1,j}\right)$. For each gate a, party P_j receives, for each other party P_i, the authentication vectors $\left([T_a]_j, \mathfrak{Y}_a^{ji}\right)$ and the check vectors $\left(\mathfrak{B}_a^{ij}, \mathfrak{C}_a^{ij}\right)$. For each output wire δ, if P_j is to learn that output, they receive mask m_δ.

In the next section, we describe the MPC protocol we will use to compute f_{GRBL}.

6.2 MPC with Linear Round Complexity in d and κ and Asynchronous Fallback

To achieve security in both synchronous and asynchronous networks, we want to compute f_{GRBL} using the compiler from [11]. We recall the construction and its security guarantees below.

Protocol $\Pi^{t_s,t_a}_{\mathsf{HMPC}}(\Pi_1, \Pi_2)$

We describe the protocol from the point of view of party P_j holding input b_j.

```
1: $v_j := \Pi_1(b_j)$;
2: if $v_j \neq \bot$ then output $v_j$ and terminate;
3: end if
4: $y_j := \Pi_2(b_j)$;
5: output $y_j$ and terminate;
```

Lemma 5 ([11], Theorem 2). *Assume protocol Π_1 achieves the following security guarantees.*

- *When run over a synchronous network: t_s-security.*
- *When run over an asynchronous network: t_a-unanimous output, t_a-weak termination and $(n - t_s)$-output quality.*

Furthermore, assume protocol Π_2 achieves the following security guarantees.

- *When run over an asynchronous network: t_a-security with $(n - t_a)$-output quality.*

Then, assuming $t_a \leq t_s$ and $t_a + 2t_s < n$, protocol $\Pi^{t_s,t_a}_{\mathsf{HMPC}}(\Pi_1, \Pi_2)$ achieves the following security guarantees.

1. *When run over a synchronous network: t_s-security.*
2. *When run over an asynchronous network: t_a-security and $(n - t_s)$-output quality.*

We provide sub-protocols Π_1, Π_2 with the security guarantees required by Lemma 5, and that in addition 1) securely evaluate *boolean* circuits, and 2) require $O(d)$ communication rounds.

We take Π_1 to be the synchronous protocol $\Pi^{t_a,t_s}_{\mathsf{SMPC}}$ of [11, Section 4.5], which requires $O(d)$ rounds; it is the only known synchronous protocol to date providing the necessary security guarantees.

However, we cannot use $\Pi^{t_s,t_a}_{\mathsf{SMPC}}$ in a black-box manner, since it evaluates arithmetic circuits defined over "large" fields ($\#\mathbb{F}_q > n$), while in our construction it is natural to represent the *boolean* function f_{GRBL} as a *boolean* circuit. One solution is to embed the boolean circuit into a larger field through the inclusion map $i : \mathbb{F}_2 \to \mathbb{F}_q$ and to represent `NAND` gates with arithmetic gates computing $a(x, y) := 1 - xy$ (it is straightforward to verify that $i \circ a = a \circ i$).

To keep actively corrupted parties from giving inputs in $\mathbb{F}_q \backslash \{0, 1\}$, a checking mechanism has to be put into place. The high level idea of $\Pi^{t_s,t_a}_{\mathsf{SMPC}}$ is that the inputs of each party are encrypted using an additively-homomorphic threshold encryption scheme. To ensure correctness, after broadcasting their encrypted inputs, parties must prove (in ZK) knowledge for the corresponding plaintext. *In addition, we require parties to prove in* ZK *that the plaintext lies in* $\{0, 1\}$.

The protocol then follows the gate-by-gate paradigm, with additional interaction required to evaluate multiplication gates. After the circuit is evaluated, parties reconstruct the outputs using threshold decryption. A security proof can be obtained as for [11, Theorem 1] with minor changes.

We take Π_2 to be the modified version (by Coretti et al. [17]) of protocol π_{BKR} by Ben-Or et al. [8], which evaluates boolean circuits and requires $O(d)$ rounds. Security is proven in [17, Lemma 2].

Lemma 5, with these choices of Π_1 and Π_2, yields the following corollary.

Corollary 3. *Assume $t_a \leq t_s$ and $t_a + 2t_s < n$. There exists a protocol $\Pi^{t_s,t_a}_{\mathsf{HMPC}}$ evaluating boolean circuits and requiring $O(d)$ communication rounds achieving the following security guarantees.*

- *When run over a synchronous network: t_s-security.*
- *When run over an asynchronous network: t_a-security and $(n - t_s)$-output quality.*

Our modification of Protocol $\Pi_{\mathsf{SMPC}}^{t_a,t_a}$ [11], used as Π_1 in compiler $\Pi_{\mathsf{HMPC}}^{t_a,t_s}$, requires black-box access to:

(i) a Byzantine agreement sub-protocol that is t_s-secure when run over a synchronous network, and t_a-secure when run over an asynchronous network;
(ii) a broadcast sub-protocol that is t_s-secure when run over a synchronous network, and t_a-weakly valid when run over an asynchronous network.

At the time of [11], the only known protocols with these guarantees required $O(n)$ rounds,[10] resulting in $O(n)$ round-complexity of the MPC protocol.

In Section G, we present a broadcast protocol $\Pi_{\mathsf{SBC}}^{t_a,t_s}\left(\Pi_{\mathsf{SBA}}^{t_a,t_s}\right)$ running in a fixed number of rounds that is weakly valid in asynchronous networks. Our solution is inspired by a synchronous construction that obtains BC from BA, but requires some modifications to achieve security guarantees in asynchronous networks.

Combining this with results from previous sections, we obtain an MPC protocol running in $O(\kappa)$ rounds with respect to n. More specifically,

- Lemma 1 (or Lemma 9), Lemma 2, and Lemma 4, guarantee protocol $\Pi_{\mathsf{HBA}}^{t_a,t_s}\left(\Pi_{\mathsf{SBA}}^{t_a,t_s}, \Pi_{\mathsf{ABA}}^{t_a,t_s}\right)$ from Section 5 (which runs in $O(\kappa)$ rounds with respect to the number of parties n) achieves the security guarantees (i);
- Lemma 1, (or Lemma 9), Lemma 2, and Lemma 14, guarantee protocol $\Pi_{\mathsf{SBC}}^{t_a,t_s}\left(\Pi_{\mathsf{SBA}}^{t_a,t_s}\right)$ from Section G (that also runs in $O(\kappa)$ rounds with respect to the number of parties n) achieves the security guarantees (ii).

Combining this with Corollary 3, we obtain the following corollary.

Corollary 4. *Assume $t_a \leq t_s$ and $t_a + 2t_s < n$. There exists a* MPC *protocol with the following properties.*

- *When run over a synchronous network: t_s-security.*
- *When run over an asynchronous network: t_a-security and $(n - t_s)$-output quality.*
- *If the network is synchronous, runs in $O(\kappa)$ rounds.*
- *Runs in $O(d)$ rounds.*

Recall that the security guarantees of Corollary 4 are optimal ([11, Theorems 3, 4]).

6.3 Protocol Description

We now present our fully constant-round MPC protocol that is 1) t_s-secure if the network is synchronous, and 2) t_a-secure with $(n - t_s)$-output quality if the network is asynchronous. The construction, that we already discussed, consists of three steps.

- (Parties jointly) use an MPC protocol with the properties of Corollary 4 to compute function f_{GRBL}.
- (Each party) encrypts the authenticated shares of the entries of each gate of circ_g received as output of f_{GRBL} (the keys are also part of the output). They send the resulting ciphertexts to all parties.
- (Each party) evaluates the circuit locally: given two (masked) inputs to a gate and the corresponding keys, they decrypt the received shares of the corresponding entry of the gate, recovering the (masked) output value and the corresponding key. They do this until all gates are evaluated. Finally, they unmask the accessible outputs.

[10] Respectively, the BA protocol and the adaptation of Dolev-Strong broadcast from [9].

A phase indicator ϕ guarantees that, if the network is asynchronous, parties do not terminate before sending the encryptions of their shares to other parties. Security is discussed in Section H.

Protocol $\Pi^{t_s}_{\textsf{CR-HMPC}}\left(\Pi^{t_s,t_a}_{\textsf{HMPC}}\right)$

We describe the protocol from the point of view of party P_j holding input b_j. For each gate a of $\textsf{circ}_g$ set $\textsf{evaluated}_a := \texttt{false}$. Set $\phi_j := 0$.

Step 1. Run $\Pi^{t_s,t_a}_{\textsf{HMPC}}\left(\textsf{circ}_{f_{\textsf{GRBL}}}; \textsf{circ}_g; b_j\right)$, receiving as output:

- for each wire γ of $\textsf{circ}_g$, the sub-keys $\left(K^{0,j}_\gamma, K^{1,j}_\gamma\right)$;
- for each gate a of $\textsf{circ}_g$ and party P_j, the authentication vectors $\left([T_a]_j, \mathfrak{Y}^{ji}_a\right)$ and the check vectors $\left(\mathfrak{B}^{ij}_a, \mathfrak{C}^{ij}_a\right)$;
- for each input wire ω of $\textsf{circ}_g$, the masked input value z_ω and the corresponding key $K^{z_\omega}_\omega$.
- for each output wire δ of $\textsf{circ}_g$, if P_j is entitled to learn that output, the mask m_δ.

Step 2. For each gate a of $\textsf{circ}_g$ with input wires α, β and output wire γ, do:

- for each $(x, y) \in \{0,1\}^2$, encrypt the authenticated share of the corresponding entry of T_a, namely $c^{xy,j}_a := \textsf{Enc}_{K^{x,j}_\alpha, K^{y,j}_\beta}\left([t^{xy}_a]_j, \mathfrak{y}^{xy,ji}_a\right)$;
- send $C^j_a := \left(c^{00,j}_a, c^{01,j}_a, c^{10,j}_a, c^{11,j}_a\right)$ to all parties.

Then, set $\phi_j := 1$.

Step 3. If $\phi_j = 1$, whenever a ciphertext is received, for each gate a of $\textsf{circ}_g$ with input wires α, β and output wire γ, if the masked input values z_α, z_β and the corresponding key (vectors) $K^{z_\alpha}_\alpha, K^{z_\beta}_\beta$ are known, do:

- For ciphertext C^i_a, set $\left([t^{z_\alpha z_\beta}_a]_i, \mathfrak{y}^{z_\alpha z_\beta, ij}_a\right) := \textsf{Dec}_{K^{z_\alpha,i}_\alpha, K^{z_\beta,i}_\beta}\left(c^{z_\alpha z_\beta, i}_a\right)$. If the decryption is successful and if $\mathfrak{c}^{z_\alpha z_\beta, ij}_a = [t^{z_\alpha z_\beta}_a]_i + \mathfrak{b}^{z_\alpha z_\beta, ij}_a \mathfrak{y}^{z_\alpha z_\beta, ij}_a$, then the i-th shares of z_γ and $K^{z_\gamma}_\gamma$ are recovered.
- If at least $t_s + 1$ shares have been recovered, reconstruct z_γ and $K^{z_\gamma}_\gamma$ and set $\textsf{evaluated}_a := \texttt{true}$.

When all gates are evaluated, compute $b_\omega := z_\omega \oplus m_\omega$ for all accessible output wires ω. Output bits b_ω and terminate.

Lemma 6. *Suppose $t_a \le t_s$ and $t_a + 2t_s < n$, and assume protocol $\Pi^{t_s,t_a}_{\textsf{HMPC}}$ achieves the security guarantees of* Corollary 4. *Then, protocol $\Pi^{t_s,t_a}_{\textsf{CR-HMPC}}\left(\Pi^{t_s,t_a}_{\textsf{HMPC}}\right)$ achieves the same security guarantees, and requires a number of rounds independent of the circuit depth of the function to be evaluated, when the network is synchronous.*

References

[1] Ittai Abraham, Danny Dolev, and Joseph Y. Halpern. An almost-surely terminating polynomial protocol forasynchronous byzantine agreement with optimal resilience. In Rida A. Bazzi and Boaz Patt-Shamir, editors, *27th ACM PODC*, pages 405–414. ACM, August 2008.

[2] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Maofan Yin. Sync HotStuff: Simple and practical synchronous state machine replication. Cryptology ePrint Archive, Report 2019/270, 2019. `https://eprint.iacr.org/2019/270`.

[3] Prabhanjan Ananth, Arka Rai Choudhuri, Aarushi Goel, and Abhishek Jain. Two round information-theoretic MPC with malicious security. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 532–561. Springer, Heidelberg, May 2019.

[4] Judit Bar-Ilan and Donald Beaver. Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In Piotr Rudnicki, editor, *8th ACM PODC*, pages 201–209. ACM, August 1989.

[5] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *22nd ACM STOC*, pages 503–513. ACM Press, May 1990.

[6] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 784–796, 2012.

[7] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th ACM STOC*, pages 1–10. ACM Press, May 1988.

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

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

[10] Erica Blum, Jonathan Katz, and Julian Loss. Network-agnostic state machine replication. Cryptology ePrint Archive, Report 2020/142, 2020. `https://eprint.iacr.org/2020/142`.

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

[12] Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in Constantinople: Practical asynchronous byzantine agreement using cryptography. *Journal of Cryptology*, 18(3):219–246, July 2005.

[13] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[14] Ran Canetti and Tal Rabin. Fast asynchronous byzantine agreement with optimal resilience. In *25th ACM STOC*, pages 42–51. ACM Press, May 1993.

[15] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (abstract) (informal contribution). In Carl Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, page 462. Springer, Heidelberg, August 1988.

[16] Ran Cohen, Sandro Coretti, Juan A. Garay, and Vassilis Zikas. Probabilistic termination and composability of cryptographic protocols. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 240–269. Springer, Heidelberg, August 2016.

[17] Sandro Coretti, Juan Garay, Martin Hirt, and Vassilis Zikas. Constant-round asynchronous multi-party computation based on one-way functions. In *Proceedings, Part II, of the 22nd International Conference on Advances in Cryptology — ASIACRYPT 2016 - Volume 10032*, page 998–1021, Berlin, Heidelberg, 2016. Springer-Verlag.

[18] Ronald Cramer, Ivan Damgård, Stefan Dziembowski, Martin Hirt, and Tal Rabin. Efficient multiparty computations secure against an adaptive adversary. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 311–326. Springer, Heidelberg, May 1999.

[19] Ronald Cramer, Ivan Damgård, and Ueli M. Maurer. General secure multi-party computation from any linear secret-sharing scheme. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 316–334. Springer, Heidelberg, May 2000.

[20] Ivan Damgård and Yuval Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In *Annual International Cryptology Conference*, pages 378–394. Springer, 2005.

[21] Ivan Damgård and Jesper Buus Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 247–264. Springer, Heidelberg, August 2003.

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

[23] Paul Feldman and Silvio Micali. Optimal algorithms for byzantine agreement. In *20th ACM STOC*, pages 148–161. ACM Press, May 1988.

[24] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.

[25] Matthias Fitzi, Martin Hirt, and Ueli M. Maurer. Trading correctness for privacy in unconditional multi-party computation (extended abstract). In Hugo Krawczyk, editor, *CRYPTO'98*, volume 1462 of *LNCS*, pages 121–136. Springer, Heidelberg, August 1998.

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

[27] Yue Guo, Rafael Pass, and Elaine Shi. Synchronous, with a chance of partition tolerance. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 499–529. Springer, Heidelberg, August 2019.

[28] Martin Hirt and Ueli M. Maurer. Robustness for free in unconditional multi-party computation. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 101–118. Springer, Heidelberg, August 2001.

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

[30] Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for byzantine agreement. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 445–462. Springer, Heidelberg, August 2006.

[31] Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for byzantine agreement. In *Annual International Cryptology Conference*, pages 445–462. Springer, 2006.

[32] Yehuda Lindell, Anna Lysyanskaya, and Tal Rabin. Sequential composition of protocols without simultaneous termination. In Aleta Ricciardi, editor, *21st ACM PODC*, pages 203–212. ACM, July 2002.

[33] Shengyun Liu, Paolo Viotti, Christian Cachin, Vivien Quéma, and Marko Vukolić. Xft: Practical fault tolerance beyond crashes. In *12th USENIX Symposium on Operating Systems Design and Implementation*, pages 485–500, 2016.

[34] Chen-Da Liu-Zhang, Julian Loss, Ueli Maurer, Tal Moran, and Daniel Tschudi. MPC with synchronous security and asynchronous responsiveness. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 92–119. Springer, Heidelberg, December 2020.

[35] Julian Loss and Tal Moran. Combining asynchronous and synchronous byzantine agreement: The best of both worlds. Cryptology ePrint Archive, Report 2018/235, 2018. `https://eprint.iacr.org/2018/235`.

[36] Dahlia Malkhi, Kartik Nayak, and Ling Ren. Flexible byzantine fault tolerance. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1041–1053, 2019.

[37] Silvio Micali. Very simple and efficient byzantine agreement. In Christos H. Papadimitriou, editor, *ITCS 2017*, volume 4266, pages 6:1–6:1, 67, January 2017. LIPIcs.

[38] Satoshi Nakamoto. A peer-to-peer electronic cash system. 2008.

[39] Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 91. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[40] Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 3–33. Springer, Heidelberg, April / May 2018.

[41] Arpita Patra, Ashish Choudhary, and C. Pandu Rangan. Simple and efficient asynchronous byzantine agreement with optimal resilience. In Srikanta Tirthapura and Lorenzo Alvisi, editors, *28th ACM PODC*, pages 92–101. ACM, August 2009.

[42] Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM (JACM)*, 27(2):228–234, 1980.

[43] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 73–85, 1989.

[44] Robert Shostak, Marshall Pease, and L Lamport. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.

[45] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164. IEEE Computer Society Press, November 1982.

[46] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.

Appendix

A Additional Definitions

Symmetric-Key Encryption. We recall the definition of a symmetric encryption scheme.

Definition 7. *A symmetric encryption scheme is a triple of algorithms* $(\mathsf{Enc}, \mathsf{Dec}, \mathsf{Kgn})$ *such that:*

- *the key generation algorithm* Kgn *outputs a secret key* $K \in \mathcal{K}$*;*
- *given a secret key* $K \in \mathcal{K}$ *and a plaintext* $m \in \{0,1\}^*$*, the encryption algorithm* Enc *outputs a ciphertext* $\mathcal{C} \ni c := \mathsf{Enc}_K(m)$*;*
- *given a ciphertext* $c \in \mathcal{C}$ *and a secret key* $K \in \mathcal{K}$*, the decryption algorithm* Dec *outputs* $\mathsf{Dec}_K(c) \in \{0,1\}^*$*;*
- $\mathsf{Dec}_K(\mathsf{Enc}_K(m)) = m$ *for all* $m \in \{0,1\}^*$ *and* $K \in \mathcal{K}$*.*

In a dual key encryption scheme, two keys K_1, K_2 *are needed to encrypt and decrypt. The semantics are otherwise unchanged.*

Secret-Sharing. A secret-sharing scheme allows a dealer D to distribute a secret s among a set $\mathcal{P}$ of n parties, so that only certain qualified subsets of parties can reconstruct the secret. Other subsets should obtain no information about the secret. A secret-sharing scheme is specified by its *access structure* $\Gamma \subseteq 2^{\mathcal{P}}$: the collection of the qualified subsets of parties.

Definition 8. *A secret-sharing scheme for access structure* Γ *is a pair* $(\mathsf{Share}, \mathsf{Reconstruct})$ *of protocols with the following properties.*

- *After* Share (s)*, there is a unique value* s' *that can be reconstructed, and* $s' = s$ *if the dealer is honest. Furthermore, any subset of parties* $S \in \Gamma$ *can execute* $\mathsf{Reconstruct}$ *to reconstruct* s*.*
- *After* Share (s)*, any subset of parties* $S \notin \Gamma$ *obtains no information about* s*.*

We are interested in t-out-of-n secret-sharing schemes, that is, secret sharing schemes where $\Gamma := \{S \in 2^{\mathcal{P}} : \#S \geq t\}$.

B Gradecast with Asynchronous Weak Validity

We present, using slightly different notation, a 4-round gradecast protocol by Katz et al. [31] and explicitly show that their construction achieves *t-weak graded validity* (q.v. Definition 5) for all $t < n/2$ when the network is asynchronous.

Protocol Π^t_{GBC}

Unless specified, we describe the protocol from the point of view of party P_j.

Round 1 - Sender P^*. Send $(v^*, \mathsf{Sgn}(v^*, \mathtt{sk}^*))$ to all parties.

Round 1. Let (v_j, σ_j) be the first message received from P^*. If $\mathsf{Vfy}(v_j, \sigma_j, \mathtt{pk}^*) = 1$, forward (v_j, σ_j) to all parties in Round 2. In all other cases, set $v_j := \bot$ (and do not send any message in Round 2).

Round 2. Let (v_{ij}, σ_{ij}) be the first message received from P_i. If there is $i \in \{1, \dots, n\}$ such that $v_{ij} \neq v_j$, set $v_j := \bot$. If $v_j \neq \bot$, send message $(v_j, \mathsf{Sgn}(v_j, \mathtt{sk}_j))$ to all parties in Round 3.

Round 3. Upon receiving at least $t+1$ valid messages (b, σ_{j_k}) (i.e. such that $\mathsf{Vfy}(b, \sigma_{j_k}, \mathtt{pk}_{j_k}) = 1$) from distinct parties on the same bit b, set $\Sigma := \{\sigma_{j_1}, \dots, \sigma_{j_{t+1}}\}$, send (b, Σ) to all parties in Round 4, and output $(b, 2)$.

Round 4. If no output has been generated, upon receiving a message (b', Σ') such that Σ' is a $(t+1)$-

certificate for b', output $(b', 1)$. If no output has been generated and no certificate is received, output $(\perp, 0)$. Terminate.

Lemma 7. *Assume the network is synchronous. If an honest P_j sends message $(v, \mathsf{Sgn}(v, \mathtt{sk}_j))$ in* Round 3*, no honest P_i $(i \neq j)$ sends message $(1 - v, \mathsf{Sgn}(1 - v, \mathtt{sk}_i))$ in* Round 3*.*

Proof. We prove the lemma by contradiction. Below, all bits are to be intended as validly signed. Assume honest parties P_j and P_i send bits v and $1 - v$ respectively in Round 3. Observe that an honest party sends a bit in round 3 if and only if this is the bit they received from the sender P^*. This means P^* sent bit v to P_j and $1 - v$ to P_i in Round 1. Therefore, P_j and P_i forward bits v and $1 - v$ respectively to all honest parties in Round 2. In particular, both P_j and P_i see conflicting messages in Round 2 and do not send a message in Round 3. This is a contradiction. □

Lemma 8. *Assume $t < n/2$. Protocol Π^t_{GBC} achieves the following security guarantees.*

- *When run over a synchronous network: t-graded validity and t-graded consistency.*
- *When run over an asynchronous network: t-weak graded validity.*

Proof. We prove each claim separately.

[**graded validity**] If the sender P^* is honest, in Round 1 honest parties receive $(v^*, \sigma^* = \mathsf{Sgn}(v^*, \mathtt{sk}^*))$. Therefore, each honest party checks that $\mathsf{Vfy}(v^*, \sigma^*, \mathtt{pk}^*) = 1$ and forwards (v, σ^*) to all honest parties in Round 2. Since dishonest parties cannot forge an honest sender's signature, no honest party receives contradicting messages in Round 2. Hence, all honest parties send $(v^*, \mathsf{Sgn}(v^*, \mathtt{sk}_j))$ to all parties in Round 3. In conclusion, in Round 3, each honest party P_j receives at least $n - t > t$ messages $(v^*, \mathsf{Sgn}(v^*, \mathtt{sk}_i))$ for distinct $i \in \{1, \ldots, n\}$, and they output $(v, 2)$ (and ignore Round 4).

[**graded consistency- b.**] Assume an honest party P_j outputs $(v, 2)$. This means they received $t + 1$ messages $(v, \mathsf{Sgn}(v, \mathtt{sk}_i))$ for distinct $i \in \{1, \ldots, n\}$ in Round 3. Hence, at least one honest party P_k sent $(v, \mathsf{Sgn}(v, \mathtt{sk}_k))$ in Round 3. By Lemma 7, no honest party P_l sends $(v, \mathsf{Sgn}(1 - v, \mathtt{sk}_l))$ in Round 3, and each honest party receives at most $t < n - t$ validly signed bits $1 - v$ in Round 3, so that no honest party outputs $(1 - v, 2)$ in Round 3. Let P_i be an honest party. If P_i does not output $(v, 2)$ in Round 3, since the network is synchronous, they receive a valid certificate on v (from P_j) in Round 4, and they output $(v, 1)$.

[**graded consistency-a.**] We already showed that if some honest party outputs $(v, 2)$, then each honest party outputs either $(v, 2)$ or $(v, 1)$. Suppose an honest party P_j outputs $(v, 1)$ (in Round 4). This means they received a $(t + 1)$-valid certificate Σ on v in Round 4. Certificate Σ contains at least $t + 1$ valid signatures on v, so that at least one honest party sent the validly signed bit v at the beginning of Round 3. By Lemma 7, no honest party signs and sends message $1 - v$ at the beginning of Round 3. Therefore, each honest party receives at most t validly signed bits $1 - v$ in Round 3, and does not output $(1 - v, 2)$. In particular, no honest party sends a $(t + 1)$-certificate on $1 - v$ in Round 4. Since corrupted parties cannot forge more than t signatures on $1 - v$, no valid certificate on $1 - v$ can be produced by the adversary, and no honest party receives a $(t + 1)$-valid certificate on $1 - v$ in Round 4. Hence, no honest player outputs $(1 - v, 1)$.

[**weak graded validity**] If the sender P^* is honest and has input v^*, no honest party receives the validly signed bit $1 - v^*$ in Round 1. In particular, no honest party ever signs bit $1 - v$ throughout the protocol. Therefore, each honest party receives at most t validly signed bits $1 - v$ in Round 3, and does not output $(1 - v, 2)$ and send a certificate on $1 - v$ in Round 4. Since dishonest parties cannot produce a $(t + 1)$-certificate for bit $1 - v$ (they can produce at most t valid signatures on $1 - v$), no honest party receives a $(t + 1)$-certificate on $1 - v$ in Round 4: they do not output $(1 - v, 1)$. □

C Proof of Lemma 1

Assume that that at most t_s parties are corrupted in an execution of $\Pi_{\mathsf{WC}}^{t_a,t_s}\left(\Pi_{\mathsf{GC}}^{\max\{t_a,t_s\}}\right)$ over a synchronous network.

[**liveness**] synchrony of the network and t_s-graded validity of protocol $\Pi_{\mathsf{GC}}^{\max\{t_a,t_s\}}$ guarantee that each honesty party P_j sets $b_{ij} := \Pi_{\mathsf{GC}}^{\max\{t_a,t_s\}}(i) = (v_i, 2)$ each time party P_i is honest. Therefore, $\#(S_j^v \sqcup S_j^{1-v}) \geq n - t_s$, so that P_j sets $b_j \in \{0, 1, \bot\}$ during output determination and does not output $\top$. This proves t_s-liveness.

[**validity**] Suppose that all honest parties hold the same input v. synchrony of the network and t_s-graded validity of protocol $\Pi_{\mathsf{GC}}^{\max\{t_a,t_s\}}$ guarantee that each honesty party P_j sets $b_{ij} := \Pi_{\mathsf{GC}}^{\max\{t_a,t_s\}}(i) = (v, 2)$ each time party P_i is honest. Therefore, $\#S_j^v \geq n - t_s$ and party P_j outputs v. It is worth noting that if both $\#S_j^v \geq n - t_s$ and $\#S_j^{1-v} \geq n - t_s$, then $n \geq \#(S_j^v \sqcup S_j^{1-v}) \geq 2n - 2t_s > n$, which is absurd. This proves t_s-validity.

[**weak consistency**] Suppose an honest party P_j outputs $v \in \{0, 1\}$. There are two possibilities. The first is that

$$\begin{cases} \#S_j^v \geq n - t_s - t_a \\ \#(S_j^{1-v} \sqcup U_j^{1-v}) \leq t_a. \end{cases} \tag{1}$$

If P_i is another honest party, then synchrony of the network and t_s-graded consistency of $\Pi_{\mathsf{GBC}}^{\max\{t_a,t_s\}}$ guarantee that $\#S_i^{1-v} \leq t_a < n - t_s - t_a \leq n - t_s$. If this was not the case, by t_s-graded consistency of $\Pi_{\mathsf{GBC}}^{\max\{t_a,t_s\}}$ we would have $\#(S_j^{1-v} \sqcup U_j^{1-v}) > t_a$, which is a contradiction. Therefore, party P_i does not output $1 - v$. The second case is that $\#S_j^v \geq n - t_s$. In this case (reasoning as above), for an honest player P_i

$$\begin{cases} \#(S_i^v \sqcup U_i^v) \geq n - t_s > 2t_a \geq t_a \\ \#S_i^{1-v} < n - t_s \end{cases} \tag{2}$$

so that P_i does not output $1 - v$. This proves t_s-weak consistency.

Assume that that at most t_a parties are corrupted in an execution of $\Pi_{\mathsf{WC}}^{t_a,t_s}\left(\Pi_{\mathsf{GC}}^{\max\{t_a,t_s\}}\right)$ over an asynchronous network.

[**weak validity**] Assume all honest parties hold the same input v. Suppose an honest party P_j does not output $\top$. This means $\#(S_j^v \sqcup S_j^{1-v}) \geq n - t_s$. By t_a-weak graded validity of protocol $\Pi_{\mathsf{GC}}^{\max\{t_a,t_s\}}$, party P_j sets $b_{ij} := \Pi_{\mathsf{GC}}^{\max\{t_a,t_s\}}(i) \in \{v, \top\}$ for each honest party P_i. Therefore, $\#S_j^{1-v} \leq t_a < n - t_s - t_a \leq n - t_s$ (so that party P_j does not output $1 - v$),but also

$$\begin{cases} \#S_j^v \geq n - t_s - \#S_j^{1-v} \geq n - t_s - t_a \\ \#(S_j^{1-v} \sqcup U_j^{1-v}) \leq t_a \end{cases} \tag{3}$$

so that party P_j outputs v. This proves t_a-weak validity and concludes the proof of the lemma.

D A Simpler Weak-Consensus with Asynchronous Weak Validity for $t_a + 2t_s < n$ and $t_a \leq t_s$

We show a simple 3-round construction for a weak consensus protocol that is 1) t_s-secure in a synchronous network, and 2) t_a-weakly valid in an asynchronous network, under the stronger assumptions that $t_a + 2t_s < n$, $t_a \leq t_s$ (these assumptions are optimal for BA with full asynchronous fallback [9]). The public key infrastructure available allows parties to forward cryptographic evidence (in the form of digital signatures) that they received a given message from other parties by appropriately combining this evidence to generate what we refer to as *certificates* (see e.g. [29]). An ℓ-certificate on a bit b is simply a concatenation of at least ℓ valid signatures on b from distinct parties.

Protocol $\Pi_{\mathsf{WC}}^{t_a,t_s}$

We describe the protocol from the point of view of party P_j holding input v_j.

Initialization step. Set $b_j := \top$, $S_j := \varnothing$.

Round 1. Send message $(v_j, \mathsf{Sgn}(v_j, \mathtt{sk}_j))$ to all parties. Upon receiving (the first) message $m_{ij} = (v_{ij}, \sigma_{ij})$ from party P_i, if $\mathsf{Vfy}(v_{ij}, \sigma_{ij}, \mathtt{pk}_i) = 1$, set $S_j := S_j \cup \{m_{ij}\}$.

Round 2. If $\#S_j \geq n - t_s$, set $b_j = \bot$. If there is $b \in \{0,1\}$ such that $\#S_j^b := \{(v,\sigma) \in S_j \,:\, v = b\} \geq n - t_s - t_a$, set $b_j := b$ and send S_j^b to all parties.

Round 3. If $b_j \in \{0,1\}$, upon receiving an $(n - t_s - t_a)$-certificate on $1 - b_j$ from any party P_i, set $b_j = \bot$.

Output. Output b_j;

Lemma 9. *Assume $t_a + 2t_s < n$ and $t_a \leq t_s$. Protocol $\Pi_{\mathsf{WC}}^{t_a,t_s}$ achieves the following security guarantees.*

- *When run over a synchronous network: t_s-liveness, t_s-validity, and t_s-weak consistency.*
- *When run over an asynchronous network: t_a-weak validity.*

Proof. Assume that at most t_s parties are corrupted in an execution of $\Pi_{\mathsf{WC}}^{t_a,t_s}$ over a synchronous network.

[**liveness**] Each honest party P_j sends message $(v_j, \mathsf{Sgn}(v_j, \mathtt{pk}_j))$ to all parties at in Round 1. synchrony of the network guarantees all these messages are delivered within the round. It follows that, in Round 2, $\#S_j \geq n - t_s$ for each honest party P_j, so that P_j sets $b_j = \bot$. This proves t_s-liveness, as b_j is never set to $\top$.

[**validity**] Assume each honest party holds the same input $v \in \{0,1\}$. In Round 1, each honest P_j sends message $(v_j, \mathsf{Sgn}(v_j, \mathtt{pk}_j))$ to all parties. synchrony of the network guarantees that $\#S_j^v \geq n - t_s \geq n - t_s - t_a$ for each honest party P_j, so that P_j sets $b_j = v$ in Round 2. Notice that $\#S_j^{1-v} \leq t_s < n - t_s - t_a$. No honest party signs bit $1 - v$ at any point in the execution of the protocol, and the adversary cannot forge signatures on behalf of honest parties. Together with $t_s < n - t_s - t_a$, this implies that no $(n - t_s - t_a)$-certificate on bit $1 - v$ can be produced by corrupted parties, so that no honest party P_j sets $b_j = \bot$ in Round 3. In conclusion, each honest party P_j outputs $b_j = v$. This proves t_s-validity.

[**weak consistency**] Assume an honest party P_j outputs v. This means P_j sets $b_j = v$ in Round 2, and sends a $(n - t_s - t_a)$-certificate on v to all parties in Round 3. synchrony of the network guarantees that this certificate is delivered to all honest parties by the end of the round. In conclusion, no honest party outputs $1 - v$. This proves t_s-weak consistency.

Assume that that at most t_a parties are corrupted in an execution of $\Pi_{\mathsf{WC}}^{t_a,t_s}$ over an asynchronous network.

[**weak validity**] Assume each honest party holds the same input $v \in \{0,1\}$ and assume an honest party P_j does not output $\top$. This means $\#S_j \geq n - t_s$. Notice that $S_j = S_j^v \sqcup S_j^{1-v}$. The adversary cannot forge honest parties' signatures, which guarantees $\#S_j^{1-v} \leq t_a$; this implies $\#S_j^v \geq \#S_j - t_a \geq n - t_s - t_a$, so that P_j sets $b_j = v$ in Round 2. The assumption $t_a \leq t_s$ guarantees that $t_a \leq t_s < n - t_s - t_a$, which means corrupted parties cannot produce an $(n - t_s - t_a)$-certificate on $1 - v$. In conclusion, party P_j outputs $b_j = v$ in Round 3. This proves t_a-weak validity and concludes the proof of the lemma. □

E Proof of Lemma 2

Assume that that at most t_s parties are corrupted in an execution of $\Pi_{\mathsf{SBA}}^{t_a,t_s}$ over a synchronous network.

[**liveness**] We claim that each honest party P_j inputs $b_j \in \{0,1\}$ to the execution of $\Pi_{\mathsf{WC}}^{t_a,t_s}$ in iteration k (for all k). This holds trivially for $k=1$. Suppose it holds for k. synchrony of the network guarantees that, by t_s-liveness of $\Pi_{\mathsf{WC}}^{t_a,t_s}$, $b_j \in \{0,1,\bot\}$ for each honest party P_j after running weak-consensus in iteration k. Since $\mathsf{coin}_k \in \{0,1\}$, then $b_j \in \{0,1\}$ for each honest party P_j at the end of iteration k, so that P_j inputs $b_j \in \{0,1\}$ to the execution of $\Pi_{\mathsf{WC}}^{t_a,t_s}$ in iteration $k+1$. The claim follows by induction on k. Therefore, after iteration κ, party P_j outputs $b_j \in \{0,1\}$. This proves t_s-liveness.

[**validity**] Assume each honest party P_j holds the same input $v \in \{0,1\}$. We claim that each honest party P_j inputs v to the execution of $\Pi_{\mathsf{WC}}^{t_a,t_s}$ in iteration k (for all k). This holds trivially for $k=1$. Suppose it holds for k. synchrony of the network guarantees that, by t_s-validity of $\Pi_{\mathsf{WC}}^{t_a,t_s}$, $b_j = v \in \{0,1\}$ for each honest party P_j after round the execution of weak-consensus in iteration k. Therefore, party P_j ignores the value coin_k and keeps $b_j = v$ at the end of the iteration. In conclusion, party P_j inputs v to the execution of $\Pi_{\mathsf{WC}}^{t_a,t_s}$ in iteration $k+1$. The claim follows by induction on k. Therefore, after iteration κ, party P_j outputs $b_j = v$. This proves t_s-validity.

[**consistency**] synchrony of the network guarantees that, by t_s-weak consistency of $\Pi_{\mathsf{WC}}^{t_a,t_s}$, after the execution of weak consensus in iteration k, there is $b^k \in \{0,1\}$ such that $b_j = b^k$ or $b_j = \bot$ for each honest party P_j (for all k). Since coin_k is a uniformly random bit (independent of b^k, since the adversary only learns the value coin_k after each honest party has produced output from weak consensus in iteration k), then $\mathbb{P}(\mathsf{coin}_k = b^k) = 1/2$ for all k. Furthermore, synchrony of the network guarantees that, by t_s-validity of $\Pi_{\mathsf{WC}}^{t_a,t_s}$, if $\mathsf{coin}_k = b^k$ for some k, then $b_j = b^k$ at the end of iteration k for each honest party P_j and for all $k' \geq k$ (the proof is by induction on k' as above, and we omit it).

For each positive integer k, let agree_k denote the event that there exists $b \in \{0,1\}$ such that $b_j = b$ for each honest party P_j at the end of iteration k. We denote by agree_0 the event that all honest parties hold the same input. Furthermore, let abort_k denote the event that some honest party P_j outputs $\bot$ from the execution of $\Pi_{\mathsf{WC}}^{t_a,t_s}$ in iteration k. For each $k \geq 0$ we have

$$\begin{aligned}
\mathbb{P}(\mathsf{agree}_{k+1} \,|\, \mathsf{agree}_k^c) &= \mathbb{P}\big(\mathsf{agree}_{k+1} \cap (\mathsf{abort}_{k+1} \sqcup \mathsf{abort}_{k+1}^c) \,|\, \mathsf{agree}_k^c\big) \\
&= \mathbb{P}(\mathsf{agree}_{k+1} \cap \mathsf{abort}_{k+1} \,|\, \mathsf{agree}_k^c) + \mathbb{P}(\mathsf{agree}_{k+1} \cap \mathsf{abort}_{k+1}^c \,|\, \mathsf{agree}_k^c) \\
&= \mathbb{P}(\mathsf{agree}_{k+1} \,|\, \mathsf{abort}_{k+1} \cap \mathsf{agree}_k^c)\mathbb{P}(\mathsf{abort}_{k+1}) \\
&+ \mathbb{P}(\mathsf{agree}_{k+1} \,|\, \mathsf{abort}_{k+1}^c \cap \mathsf{agree}_k^c)\mathbb{P}(\mathsf{abort}_{k+1}^c) \\
&= \mathbb{P}(\mathsf{coin}_{k+1} = b^{k+1})\mathbb{P}(\mathsf{abort}_{k+1}) + 1 \cdot \mathbb{P}(\mathsf{abort}_{k+1}^c) \\
&= \frac{1}{2}\left(\mathbb{P}(\mathsf{abort}_{k+1}) + \mathbb{P}(\mathsf{abort}_{k+1}^c)\right) + \frac{1}{2}\mathbb{P}(\mathsf{abort}_{k+1}^c) \geq \frac{1}{2}.
\end{aligned} \tag{4}$$

Notice, once again, that the above equality $\mathbb{P}(\mathsf{agree}_{k+1} \,|\, \mathsf{abort}_{k+1} \cap \mathsf{agree}_k^c) = \mathbb{P}(\mathsf{coin}_{k+1} = b^{k+1})$ holds because t_s corrupted parties alone cannot learn coin_{k+1} in advance, so that the output of honest parties in the execution of $\Pi_{\mathsf{WC}}^{t_a,t_a}$ is independent from the value of coin_{k+1} in iteration

$k+1$. The observation that $\mathsf{agree}_k^c \supseteq \mathsf{agree}_{k+1}^c$ allows us to finally estimate

$$\begin{aligned}\mathbb{P}(\mathsf{agree}_\kappa^c) &= \mathbb{P}\left(\bigcap_{k=1}^{\kappa} \mathsf{agree}_k^c\right) \\ &= \mathbb{P}\left(\mathsf{agree}_\kappa^c \;\middle|\; \bigcap_{k=1}^{\kappa-1} \mathsf{agree}_k^c\right) \mathbb{P}\left(\bigcap_{k=1}^{\kappa-1} \mathsf{agree}_k^c\right) \\ &= \prod_{k=1}^{\kappa} \mathbb{P}(\mathsf{agree}_k^c \mid \mathsf{agree}_{k-1}^c) \leq \frac{1}{2^\kappa}.\end{aligned} \tag{5}$$

This proves t_s-consistency.

Assume that that at most t_a parties are corrupted in an execution of $\Pi_{\mathsf{SBA}}^{t_a,t_s}$ over an asynchronous network.

[weak validity] Assume each honest party P_j holds the same input $v \in \{0,1\}$. We claim that each honest party P_j inputs v to the execution of $\Pi_{\mathsf{WC}}^{t_a,t_s}$ in iteration k (for all k). The claim is trivially true for $k=1$. Assume it is true for k. By t_a-weak validity of protocol $\Pi_{\mathsf{WC}}^{t_a,t_s}$, each honest party P_j outputs either v or $\top$ from $\Pi_{\mathsf{WC}}^{t_a,t_s}$ in iteration k. Therefore, each honest party P_j ignores the coin-flip value and sets $b_j = v$ at the end of iteration k, and therefore inputs $b_j = v$ to the following execution of $\Pi_{\mathsf{WC}}^{t_a,t_s}$ in iteration $k+1$. The claim follows by induction on k. In conclusion, each honest party outputs $b_j = v$ at the end of iteration κ. This proves t_a-weak validity.

F Expected Constant-Round Synchronous BA with Asynchronous Weak Validity

In this section, we describe protocol $\Pi_{\mathsf{eSBA}}^{t_a,t_s}$: an expected constant-round BA with asynchronous weak validity. This is achieved by modifying the construction $\Pi_{\mathsf{SBA}}^{t_a,t_s}$ from Section 4.1.

Assume the network is synchronous. An observation from the fixed-round protocol is that, if within an iteration any party outputs the same bit $b \in \{0,1\}$ from both protocol $\Pi_{\mathsf{WC}}^{t_a,t_s}$ and functionality $\mathcal{F}_{\mathsf{CoinFlip}}^{t_s}$, then all honest parties are in agreement on bit b (i.e. each honest party will input b to the execution of $\Pi_{\mathsf{WC}}^{t_a,t_s}$ in the following iteration). Moreover, agreement is preserved by t_s-validity of protocol $\Pi_{\mathsf{WC}}^{t_a,t_s}$ in all subsequent iterations.

To achieve early termination, the high-level (but not fully accurate) idea is that if in iteration k party P_j outputs $b \in \{0,1\}$ from both protocol $\Pi_{\mathsf{WC}}^{t_a,t_s}$ and functionality $\mathcal{F}_{\mathsf{CoinFlip}}^{t_s}$, then P_j detects that agreement has been reached. Party P_j can then output b and keep running until the first following iteration $(k + C_1)$ in which they obtain output b from functionality $\mathcal{F}_{\mathsf{CoinFlip}}^{t_s}$ once again, where P_j is sure that all parties detected agreement. At this point, P_j terminates.

This idea is not fully accurate, because when P_j terminates, there might still be parties that have not terminated. Therefore, in the following iterations, the security guarantees of $\Pi_{\mathsf{WC}}^{t_a,t_s}$ and $\mathcal{F}_{\mathsf{CoinFlip}}^{t_s}$ may be compromised, as some honest parties are no-longer participating. For example, it could be that parties do not obtain an output from $\mathcal{F}_{\mathsf{CoinFlip}}^{t_s}$ anymore.

To solve this, we follow the approach by Micali [37], and alternate invocations to functionality $\mathcal{F}_{\mathsf{CoinFlip}}^{t_s}$ with iterations in which the value of the coin is fixed to opposite bits. Each macro-iteration then consists of 3 mini-iterations, where the random coin, coin fixed to bit 0, and coin fixed to bit 1 are executed, respectively.

As soon as a party detects agreement, they output this value, but only terminate the second time the coin coincides with the output value. This happens by the end of the next large iteration: when an honest party terminates, the coin value matches their output value in one of the mini-iterations. Therefore, each honest party terminates regardless of what the values from $\Pi_{\mathsf{WC}}^{t_a,t_s}$ and $\mathcal{F}_{\mathsf{CoinFlip}}^{t_s}$ are.[11]

Protocol $\Pi_{\mathsf{eSBA}}^{t_a,t_s}\left(\Pi_{\mathsf{WC}}^{t_a,t_s}, \mathcal{F}_{\mathsf{CoinFlip}}^{t_s}\right)$

We describe the protocol from the point-of-view of party P_j holding input v_j.

Initialization step: Set $b_j := v_j$, $k := 1$, $\mathsf{ready\text{-}to\text{-}terminate}_j := 0$.

```
 1: while 1 do
 2:     $c_j := \Pi_{\mathsf{WC}}^{t_a,t_s}(b_j)$;
 3:     if $\mathsf{ready\text{-}to\text{-}terminate}_j = 0$ then $b_j := c_j$;
 4:     end if
 5:     if $k \equiv 1 \mod 3$ then $(k, \mathsf{coin}_k) := \mathcal{F}_{\mathsf{CoinFlip}}^{t_s}(k)$;
 6:     else if $k \equiv 2 \mod 3$ then $(k, \mathsf{coin}_k) := (k, 0)$;
 7:     else if $k \equiv 0 \mod 3$ then $(k, \mathsf{coin}_k) := (k, 1)$;
 8:     end if
 9:     if $b_j = \mathsf{coin}_k$ then
10:         if $\mathsf{ready\text{-}to\text{-}terminate}_j = 0$ then
11:             output $b_j$;
12:             $\mathsf{ready\text{-}to\text{-}terminate}_j := 1$;
13:         else
14:             terminate;
15:         end if
16:     end if
17:     if $b_j = \bot$ then $b_j := \mathsf{coin}_k$;
18:     else if $b_j = \top$ then $b_j := v_j$;
19:     end if
20:     if $k = \kappa$ then output $b_j$ and terminate;
21:     end if
22:     $k := k + 1$;
23: end while
```

Lemma 10. *Assume protocol $\Pi_{\mathsf{WC}}^{t_a,t_s}$ achieves the following security guarantees.*

- *When run over a synchronous network: t_s-liveness, t_s-validity, and t_s-weak consistency.*
- *When run over an asynchronous network: t_a-weak validity.*

Then, protocol $\Pi_{\mathsf{eSBA}}^{t_a,t_s}\left(\Pi_{\mathsf{WC}}^{t_a,t_s}, \mathcal{F}_{\mathsf{CoinFlip}}^{t_s}\right)$ achieves the following security guarantees with overwhelming probability.

- *When run over a synchronous network: t_s-security. Moreover, the protocol runs in $O(1)$ rounds in expectation.*
- *When run over an asynchronous network: t_a-weak validity.*

We divide the proof in a sequence of lemmas. We first argue about termination. The first lemma shows that, if an honest party generates output in iteration k, then all honest parties terminate soon after.

Lemma 11. *If the network is synchronous, and at most t_s parties are corrupted in an execution of protocol $\Pi_{\mathsf{eSBA}}^{t_a,t_s}\left(\Pi_{\mathsf{WC}}^{t_a,t_s}, \mathcal{F}_{\mathsf{CoinFlip}}^{t_s}\right)$, then the following hold.*

1. *If some honest party generates output in iteration k, all honest parties terminate at the latest in iteration $k + 6$.*
2. *If some honest party generates output in iteration k and $k \equiv 1 \mod 3$, all honest parties terminate at the latest in iteration $k + 5$.*

[11] Here, we need that parties know when $\Pi_{\mathsf{WC}}^{t_a,t_s}$ and the protocol realizing $\mathcal{F}_{\mathsf{CoinFlip}}^{t_s}$ terminate when the network is synchronous. This is the case, as these protocols terminate after a fixed constant number of rounds.

3. *If all honest parties generate output in iteration k, all honest parties terminate at the latest in iteration $k+3$.*

Proof. We prove the claims separately.

1. If $k \geq \kappa - 6$ the claim holds trivially. Assume that some honest party P_j outputs v in iteration $k < \kappa - 6$ (and therefore sets $\mathsf{ready\text{-}to\text{-}terminate}_j := 1$). If b_j is their output from the execution of protocol $\Pi_{\mathsf{WC}}^{t_a,t_s}$ in iteration k, then $b_j = v = \mathsf{coin}_k$. synchrony of the network and t_s-weak consistency of protocol $\Pi_{\mathsf{WC}}^{t_a,t_s}$ guarantee that each honest party P_i outputs $b_i \in \{v, \perp\}$ from the execution of protocol $\Pi_{\mathsf{WC}}^{t_a,t_s}$ in iteration k. Parties that output $b_i = v$ then set $b_i := b_i = v$, while parties that output $\perp$ then set $b_i := \mathsf{coin}_k = v$. Therefore, synchrony of the network and t_s-validity of protocol $\Pi_{\mathsf{WC}}^{t_a,t_s}$ guarantee that all honest parties input and output the agreed upon value v to the execution of protocol $\Pi_{\mathsf{WC}}^{t_a,t_s}$ in each iteration $k' \geq k$. Observe that $\mathsf{coin}_{k'} = v$ twice again before iteration $k+6$. The first time this happens, since $\mathsf{ready\text{-}to\text{-}terminate}_j = 1$, party P_j (and possibly other honest parties) terminates and each remaining honest party P_i generates output and sets $\mathsf{ready\text{-}to\text{-}terminate}_i := 1$. The second time, each remaining honest party P_i terminates (since $\mathsf{ready\text{-}to\text{-}terminate}_i = 1$).
2. Same as above, but since $k \equiv 1 \mod 3$ then $\mathsf{coin}_{k'}$ hits the agreed upon value v twice before iteration $k+5$.
3. Same as above, but since each honest party P_j now sets $\mathsf{ready\text{-}to\text{-}terminate}_j := 1$ in iteration k, the first time $\mathsf{coin}_{k'}$ hits the agreed upon value v (this happens at the latest for $k' = k+3$), all honest parties terminate.

□

We now compute the expected running time of the protocol. Let T be a random variable, taking values in $\mathbb{N} \cup \{\infty\}$, denoting the number of iterations in an execution of protocol $\Pi_{\mathsf{eSBA}}^{t_a,t_s}\left(\Pi_{\mathsf{WC}}^{t_a,t_s}, \mathcal{F}_{\mathsf{CoinFlip}}^{t_s}\right)$. We denote by $T = \infty$ the event that the execution of the protocol does not terminate (i.e. at least one honest party does not terminate). The expression $T \leq k$ is to be understood as "each honest party terminates in iteration k at the latest".

Lemma 12. *If the network is synchronous, and at most t_s parties are corrupted in an execution of protocol $\Pi_{\mathsf{eSBA}}^{t_a,t_s}\left(\Pi_{\mathsf{WC}}^{t_a,t_s}, \mathcal{F}_{\mathsf{CoinFlip}}^{t_s}\right)$, then $\mathbb{E}[T] \leq 12$.*

Proof. Let $k \equiv 1 \mod 3$ and suppose that at least one honest party is still running upon entering iteration k (the event $T \geq k$). Consider the following two cases.

1. Denote by $\mathcal{N}_k$ the event that in the execution of protocol $\Pi_{\mathsf{WC}}^{t_a,t_s}$ in iteration k all honest parties output $\perp$ (this means no honest party has yet generated output). If $\mathcal{N}_k$ happens, then each honest party P_j sets $b_j := \mathsf{coin}_k = b \in \{0,1\}$. Agreement on b among honest parties is preserved in each subsequent iteration by synchrony of the network and t_s-validity of protocol $\Pi_{\mathsf{eSBA}}^{t_a,t_s}$. Therefore, in the first following iteration k' in which $\mathsf{coin}_{k'} = b$ (this happens at the latest in iteration $k' = k+2$), all honest parties output b. By Lemma 11(3) all honest parties terminate at the latest in iteration $k+5$.
2. Denote by $\mathcal{S}_k$ the event that in the execution of protocol $\Pi_{\mathsf{WC}}^{t_a,t_s}$ in iteration k some honest party P_j outputs $v \neq \perp$. If $\mathcal{S}_k$ happens, then the probability that $\mathsf{coin}_k = v$ is $1/2$. The reason is that, since $k \equiv 1 \mod 3$, the value of coin_k is decided invoking functionality $\mathcal{F}_{\mathsf{CoinFlip}}^{t_s}$. Because t_s corrupted parties learn the output of $\mathcal{F}_{\mathsf{CoinFlip}}^{t_s}$ only after the execution of protocol $\Pi_{\mathsf{WC}}^{t_a,t_s}$ has terminated, the output v of party P_j in the execution of the protocol $\Pi_{\mathsf{WC}}^{t_a,t_s}$ is independent from the value coin_k. If $v = \mathsf{coin}_k$, then party P_j outputs v in iteration k, and by Lemma 11(2) all honest parties terminate at the latest in iteration $k+5$.

Clearly $\mathcal{S}_k = \mathcal{N}_k^c$, so that $\mathbb{P}(\mathcal{S}_k \sqcup \mathcal{N}_k) = 1$. The arguments above show that, for all $k \geq 1$ such that $k \equiv 1 \mod 3$,

$$\begin{aligned} \mathbb{P}(T \leq k+5 \,|\, \mathcal{S}_k \cap (T \geq k)) &= \frac{1}{2}; \\ \mathbb{P}(T \leq k+5 \,|\, \mathcal{N}_k \cap (T \geq k)) &= 1. \end{aligned} \tag{6}$$

Using these equalities, we can estimate $\mathbb{P}(T \leq k+5 \,|\, T \geq k)$ as follows.

$$\begin{aligned} \mathbb{P}(T \leq k+5 \,|\, T \geq k) &= \mathbb{P}(T \leq k+5 \,|\, \mathcal{S}_k \cap (T \geq k))\mathbb{P}(\mathcal{S}_k \,|\, T \geq k) \\ &\quad + \mathbb{P}(T \leq k+5 \,|\, \mathcal{N}_k \cap (T \geq k))\mathbb{P}(\mathcal{N}_k \,|\, T \geq k) \\ &= \frac{1}{2}\mathbb{P}(\mathcal{S}_k \,|\, T \geq k) + \mathbb{P}(\mathcal{N}_k \,|\, T \geq k) \\ &= \frac{1}{\mathbb{P}(T \geq k)}\left(\frac{1}{2}\mathbb{P}((T \geq k) \cap \mathcal{S}_k) + \mathbb{P}((T \geq k) \cap \mathcal{N}_k)\right) \\ &= \frac{1}{\mathbb{P}(T \geq k)}\left(\frac{1}{2} + \frac{1}{2}\mathbb{P}((T \geq k) \cap \mathcal{N}_k)\right) \geq \frac{1}{2}. \end{aligned} \tag{7}$$

Armed with this estimate, we can finally show

$$\begin{aligned} \mathbb{P}(T \leq k+5) &= \mathbb{P}((T \leq k+5) \cap ((T < k) \sqcup (T \geq k))) \\ &= \mathbb{P}((T \leq k+5) \cap (T < k)) \sqcup ((T \leq k+5) \cap (T \geq k))) \\ &= \mathbb{P}((T \leq k+5) \cap (T < k)) + \mathbb{P}((T \leq k+5) \cap (T \geq k)) \\ &= \mathbb{P}(T \leq k-1) + \mathbb{P}(T \geq k)\mathbb{P}(T \leq k+5 \,|\, T \geq k) \\ &\geq \mathbb{P}(T \leq k-1) + \frac{1}{2}\mathbb{P}(T \geq k) \\ &= \frac{1}{2} + \frac{1}{2}\mathbb{P}(T \leq k-1). \end{aligned} \tag{8}$$

In other words, for all $k \geq 0$ such that if $k \equiv 0 \mod 3$ we have $\mathbb{P}(T > k+6) \leq \frac{1}{2}\mathbb{P}(T > k)$. Since $k \equiv k+6 \equiv 0 \mod 3$, from this one inductively finds $\mathbb{P}(T > 6t) \leq \frac{1}{2^t}$ for all $t \geq 1$. Finally, we can estimate

$$\begin{aligned} \mathbb{E}[T] &= \sum_{t=0}^{\infty} \mathbb{P}(T > t) \\ &\leq 6\sum_{t=0}^{\infty} \frac{1}{2^t} = 12. \end{aligned} \tag{9}$$

□

In our setting, the presence of timeout κ is necessary to ensure termination when the network is asynchronous (indeed, in the absence of a timeout, the adversary could keep all honest parties from terminating by simply delaying messages). Above, we showed that when the network is synchronous the timeout is reached with negligible probability, so that if κ is large enough we can assume parties never reach the timeout. We conclude by proving security of protocol $\Pi_{\mathsf{eSBA}}^{t_a,t_s}\left(\Pi_{\mathsf{WC}}^{t_a,t_s}, \mathcal{F}_{\mathsf{CoinFlip}}^{t_s}\right)$.

Lemma 13. *Protocol $\Pi_{\mathsf{eSBA}}^{t_a,t_s}\left(\Pi_{\mathsf{WC}}^{t_a,t_s}, \mathcal{F}_{\mathsf{CoinFlip}}^{t_s}\right)$ achieves the following security guarantees with overwhelming probability.*

- *When run over a synchronous network: t_s-security.*
- *When run over an asynchronous network: t_a-weak validity.*

Proof. Below, by *iteration* we mean one of the mini-iterations (or, in other words, one iteration of the **while** cycle). Assume that at most t_s parties are corrupted and the network is synchronous.

[**liveness**] It follows immediately from synchrony of the network and t_s-liveness of protocol $\Pi_{\mathsf{WC}}^{t_a,t_s}$.

[**validity**] Assume all honest parties hold the same input v. synchrony of the network and t_s-validity of protocol $\Pi_{\mathsf{WC}}^{t_a,t_s}$ guarantee that, during the first iteration ($k = 1$), each honest party P_j sets $b_j := \Pi_{\mathsf{WC}}^{t_a,t_s}(v) = v$. Therefore, in the first following iteration in which $\mathsf{coin}_k = v$ (this happens for $k \leq 3$) party P_j outputs v. This proves t_s-validity.

[**consistency**] Assume some honest party P_j outputs v in iteration k. This means $\mathsf{coin}_k = b_j = v$. synchrony of the network and t_s-weak consistency of protocol $\Pi_{\mathsf{WC}}^{t_a,t_s}$ guarantee that $b_i \in \{b_j, \perp\}$ for each honest party P_i in iteration k. Therefore, each honest party P_i sets $b_i \in \{b_j, \mathsf{coin}_k\} = \{v\}$ in iteration k. Hence, in the first following iteration $k + C$ such that $\mathsf{coin}_{k+C} = v$ (this happens for $C \leq 3$), if party P_i has not yet produced output, they output v. This proves t_s-consistency.

Assume now that at most t_a parties are corrupted and the network is asynchronous.

[**weak validity**] Assume all honest parties hold the same input v. Suppose an honest party P_j outputs $b \neq \top$ in iteration k. This means b is the output of the execution of protocol $\Pi_{\mathsf{WC}}^{t_a,t_s}$ in iteration k. Then $b = v$ by t_a-weak validity of protocol $\Pi_{\mathsf{WC}}^{t_a,t_s}$. This proves t_a-weak validity and concludes the proof of the lemma.

G Synchronous Broadcast with Asynchronous Weak Validity

We now explain how to obtain a broadcast protocol that is t_s-secure in a synchronous network and t_a-weakly valid in an asynchronous network, starting from a BA with the same guarantees. In addition to the rounds required by the BA, our construction runs only 2 rounds. In particular, given a fixed-round BA, it yields a fixed-round broadcast protocol. The opposite construction (BA from broadcast) is shown in [9]. Together, these result completely resolve the question of equivalence of BA and broadcast with asynchronous weak validity.

The idea, well known in the synchronous model, is for the sender P^* to send their input to all parties in the first round; parties then run a Byzantine agreement protocol on the values they received to ensure consistency. However, this construction cannot be directly translated to our setting: if an honest party P_j does not receive a message from the sender P^* within the first round, then P^* could be corrupted, or the adversary might have delayed the message. In the former scenario, an easy patch would be to input a default value to the BA protocol, but this solution does not allow to achieve weak validity in the latter scenario. On the other hand, not inputting any message to the Byzantine agreement protocol fails to provide consistency if the network is synchronous.

We solve this problem by having parties run two BAs: one to agree on whether the sender behaved honestly, and one to agree on a received value. These executions can be carried out in parallel for improved round efficiency.

Let $\Pi_{\mathsf{SBA}}^{t_a,t_s}$ be a synchronous Byzantine agreement protocol (for example, our protocol with asynchronous weak validity from Section 4.2) which runs in s rounds.

Protocol $\Pi_{\mathsf{SBC}}^{t_a,t_s}\left(\Pi_{\mathsf{SBA}}^{t_a,t_s}\right)$

We describe the protocol from the point of view of party P_j.

Initialization step. Set $b_j := \top$, $\mathsf{received\text{-}input}_j := 0$.

Round 1 **(Sender).** Send message $(v^*, \mathsf{Sgn}(v^*, \mathtt{sk}^*))$ to all parties.

Round 1. Upon receiving a message (v_j, σ_j) from P^*, if $\mathsf{Vfy}(v_j, \sigma_j, \mathtt{pk}^*) = 1$, set $\mathsf{received\text{-}input}_j := 1$, $b_j := v_j$, and forward message (v_j, σ_j) to all parties in Round 2.

Round 2. If $\mathsf{received\text{-}input}_j = 0$ and $b_j = \top$, upon receiving (v'_j, σ'_j) from any party, if $\mathsf{Vfy}(v'_j, \sigma'_j, \mathtt{pk}^*) = 1$ set $b_j := v'_j$.

Round 3 **to** $3 + s$**.** Set $\mathsf{received\text{-}input}_j := \Pi_{\mathsf{SBA}}^{t_a,t_s}(\mathsf{received\text{-}input}_j)$. If $b_j \neq \top$, let $b_j := \Pi_{\mathsf{SBA}}^{t_a,t_s}(b_j)$, otherwise participate in protocol $\Pi_{\mathsf{SBA}}^{t_a,t_s}$ but do not send a message whenever supposed to share input.

Output determination. If $\mathsf{received\text{-}input}_j = 1$, output b_j. Otherwise, output $\top$.

Lemma 14. *Assume protocol $\Pi_{\mathsf{SBA}}^{t_a,t_s}$ achieves the following security guarantees.*

- *When run over a synchronous network: t_s-validity and t_s-consistency.*
- *When run over an asynchronous network: t_a-weak validity.*

Then, protocol $\Pi_{\mathsf{SBC}}^{t_a,t_s}\left(\Pi_{\mathsf{SBA}}^{t_a,t_s}\right)$ achieves the following security guarantees.

- *When run over a synchronous network: t_s-validity and t_s-consistency.*
- *When run over an asynchronous network: t_a-weak validity.*

Proof. Assume that that at most t_s parties are corrupted in an execution of $\Pi_{\mathsf{SBC}}^{t_a,t_s}\left(\Pi_{\mathsf{SBA}}^{t_a,t_s}\right)$ over a synchronous network.

[**validity**] If the sender P^* is honest, they send $(v^*, \mathsf{Sgn}(v^*, \mathtt{pk}^*))$ to all parties in round 1. synchrony of the network guarantees these messages are delivered within the round, so that each honest party P_j sets $\mathsf{received\text{-}input}_j := 1$ and $b_j := v^*$ in round 1. By t_s-validity of $\Pi_{\mathsf{SBA}}^{t_a,t_s}$, each honest party sets $\mathsf{received\text{-}input}_j := \Pi_{\mathsf{SBA}}^{t_a,t_s}(\mathsf{received\text{-}input}_j = 1) = 1$ and $b_j := \Pi_{\mathsf{SBA}}^{t_a,t_s}(b_j = v^*) = v^*$ in round $3 + s$, and outputs $b_j = v^*$ from $\Pi_{\mathsf{SBC}}^{t_a,t_s}\left(\Pi_{\mathsf{SBA}}^{t_a,t_s}\right)$. This proves t_s-validity.

[**consistency**] Assume an honest party P_j outputs $v \neq \top$. This means $\mathsf{received\text{-}input}_j$ equals 1 in round $3 + s$. Then, t_s-consistency of $\Pi_{\mathsf{SBC}}^{t_a,t_s}$ guarantees that $\mathsf{received\text{-}input}_i = 1$ in round $3 + s$ for each honest party P_i. Furthermore, t_s-validity of $\Pi_{\mathsf{SBC}}^{t_a,t_s}$ guarantees that at least one honest party P_k inputs $\mathsf{received\text{-}input}_k = 1$ to $\Pi_{\mathsf{SBC}}^{t_a,t_s}$ in round 3. This means party P_k received a validly signed message from the sender in round 1, and forwarded this message to all parties in round 2. synchrony of the network then guarantees $b_i \neq \top$ for each honest party P_i in round 3. Since each honest party provides a valid input, t_s-consistency of $\Pi_{\mathsf{SBC}}^{t_a,t_s}$ guarantees that $b_i = b_j = v$ in round $3 + s$ for each honest party P_i, so that P_i outputs v from $\Pi_{\mathsf{SBC}}^{t_a,t_s}\left(\Pi_{\mathsf{SBA}}^{t_a,t_s}\right)$. This proves t_s-consistency.

Assume that that at most t_a parties are corrupted in an execution of $\Pi_{\mathsf{SBC}}^{t_a,t_s}\left(\Pi_{\mathsf{SBA}}^{t_a,t_s}\right)$ over an asynchronous network.

[**weak validity**] Assume the sender P^* is honest and has input v^*. Up to (and including) round 3, an honest party P_j sets $b_j := v \neq \top$ only if they receive a message (v, σ) such that $\mathsf{Vfy}(v, \sigma, \mathtt{pk}^*) = 1$. Since corrupted parties cannot forge an honest sender's signature, $b_j \in \{v^*, \top\}$ in round 3 for each honest party P_j. Observe that, if $b_j = \top$ in round 3, party P_j does not send a message whenever they are supposed to share their input in $\Pi_{\mathsf{SBC}}^{t_a,t_s}$; this does not break t_a-weak validity of $\Pi_{\mathsf{SBC}}^{t_a,t_s}$, since messages can be arbitrarily delayed by the adversary. Therefore, t_a-weak validity of $\Pi_{\mathsf{SBC}}^{t_a,t_s}$ guarantees that $b_j \in \{v^*, \top\}$ in round $3 + s$ for each honest party P_j. In conclusion, each honest party P_j outputs either v^* or $\top$ from $\Pi_{\mathsf{SBC}}^{t_a,t_s}\left(\Pi_{\mathsf{SBA}}^{t_a,t_s}\right)$. This proves t_a-weak validity, and concludes the proof of the lemma. □

H Proof of Lemma 6

We sketch the proof. Assume at most t_s parties are corrupted and the network is synchronous. Then, t_s-security of $\Pi_{\mathsf{HMPC}}^{t_s,t_a}$ guarantees that each party receives the same correct output from the computation of f_{GRBL} in Step 1 (which takes into account the input of all honest parties). Therefore, each honest party encrypts their (authenticated) shares of each gate of circ_g and sends the resulting ciphertexts to all parties. synchrony of the network guarantees that each honest party receives at least $n-t_s > t_s$ valid (i.e. such that the information checking protocol succeeds) and consistent shares for each gate within one extra round. Since dishonest parties cannot forge authentication vectors, even a rushing adversary cannot compromise the reconstruction of the function table entries. Together with the masked inputs and the relative keys for each input wire, as well as the masks for the accessible output wires, the (only) reconstructed function table entry for each gate allows each honest party P_j to evaluate the garbled version of circ_g locally and recover the output. In particular, each honest party terminates.

Now, Assume at most t_a parties are corrupted and the network is asynchronous. Then, t_a-security of protocol $\Pi_{\mathsf{HMPC}}^{t_s,t_a}(\mathsf{circ}_{f_{\mathsf{GRBL}}}; \mathsf{circ}_g; b_j)$ guarantees that each honest party receives the same output (taking into account the inputs of at least $n - t_s$ honest parties) from the computation of f_{GRBL} in Step 1. Notice that if $\phi_j = 0$ (i.e. if P_j has not yet sent their encrypted shares), then party P_j does not terminate. Eventual delivery then guarantees that each honest party receives at least $n - t_a \geq n - t_s \geq t_s + 1$ valid and consistent encrypted shares of each function table entry of circ_g. Since dishonest parties cannot forge authentication vectors, each set of $t_s + 1$ valid shares identifies the same secret. Together with the masked inputs and the relative keys for each input wire, as well as the masks for the accessible output wires, the (only) reconstructed function table entry for each gate allows each honest party P_j to evaluate the garbled version of circ_g locally and recover the output. In particular, each honest party terminates.