Efficient and Adaptively Secure Asynchronous Binary Agreement via Binding Crusader Agreement

ITTAI ABRAHAM, VMware Research, Israel
NAAMA BEN-DAVID, VMware Research, USA
SRAVYA YANDAMURI, Dule University, USA

We present a new abstraction based on crusader agreement called *Binding Crusader Agreement* (BCA) for solving binary consensus in the *asynchronous* setting against an *adaptive* adversary. BCA has the validity, agreement, and termination properties of crusader agreement in addition to a new property called *binding*. Binding states that before the first non-faulty party terminates, there is a value $v \in \{0, 1\}$ such that no non-faulty party can output the value v in any continuation of the execution. We believe that reasoning about binding explicitly, as a first order goal, greatly helps algorithm design, clarity, and analysis. Using our framework, we solve several versions of asynchronous binary agreement against an adaptive adversary in a simple and modular manner that either improves or matches the efficiency of state of the art solutions. We do this via new BCA protocols, given a strong common coin, and via new Graded BCA protocols given an ϵ-good common coin. For crash failures, we reduce the expected time to terminate and we provide termination bounds that are linear in the goodness of the common coin. For Byzantine failures, we improve the expected time to terminate in the computational setting with threshold signatures, and match the state of the art in the information theoretic setting, both with a strong common coin and with an ϵ-good common coin.

1 INTRODUCTION

Reaching agreement in the presence of faults has been a cornerstone of distributed computing for over 40 years. In this paper we consider the *asynchronous* model with an *adaptive* adversary, where the adversary is allowed to delay any message between any two parties by any finite amount and can choose the delay and who to corrupt dynamically based on all the messages it has seen so far. The celebrated impossibility theorem of Fischer, Lynch, and Paterson (FLP) [19] states that any protocol that solves agreement in the presence of even one crash failure in the asynchronous model must have a non-terminating execution. The groundbreaking result of Ben-Or [4] shows that *free choice*, or *randomization*, is the key to solving Asynchronous Binary Agreement that terminates in a finite expected number of rounds.

Our main conceptual contribution is to highlight the importance in protocols design of obtaining a property we call *binding*. We believe protocols for asynchronous agreement algorithms against *adaptive* adversaries that are designed to obtain our binding property are modular and easy to prove correct.

Abstractly, a protocol obtains the binding property if no matter what the adversary does, it is forced to choose (bind to) in the present in a way that restricts all future outcomes of the protocol. As an example, we say a protocol obtains binding if there is a point in time τ (typically before any non-faulty party reveals their randomness) such that based on the execution until time τ, there is some value b (the value b is determined at time τ) such that any extension of this execution (no matter what the adversary does in the future), no non-faulty party outputs b.

Intuitively, the binding property helps protocols overcome the FLP impossibility by forcing the adaptive adversary to irrevocably choose (bind to) a value before knowing the random choices of the non-faulty parties. We believe that reasoning about the binding property explicitly, as a first order goal for algorithm design, greatly helps clarity and analysis. We note that proving security against *adaptive* adversaries is notoriously difficult and error prone. Even in the synchronous model implementing the natural ideal functionality is non-trivial [21, 22].

We use the following standard definitions to capture randomness via the abstract notion of a *common coin*, first introduced by Rabin [29]. Roughly speaking, an *ϵ-good*, *t*-unpredictable common coin needs to have two

Authors' addresses: Ittai Abraham, VMware Research, Israel, iabraham@vmware.com; Naama Ben-David, VMware Research, USA, bendavidn@vmware.com; Sravya Yandamuri, Dule University, USA, syandamu@cs.duke.edu.

properties: (1) *Unpredictability* the adversary cannot predict the coin value before at least $t + 1$ parties start participating in the protocol; and (2) *Good*: for any value $v \in \{0, 1\}$, with probability $\geq \epsilon$, all non-faulty parties output v. Many $O(1)$-good common coins are based on Verifiable Secret Sharing [1, 7, 10, 18]. A $1/2$-good common coin is called a *strong* common coin and can be implemented in the authenticated setting using secret sharing [8, 30]. A canonical example is simply having each party choose a local random coin. This is a 2^{-n}-good common coin, which we call the *local common coin*.

Our framework is based on a new abstraction that we call *Binding Crusader Agreement* (BCA). As in standard Crusader Agreement [15], this functionality is similar to agreement, but allows some parties to output a special $\perp$ value. More specifically, it guarantees (1) Validity: when all non-faulty parties have the same input, this is the only output; and (2) Agreement: no two honest parties output two distinct non-$\perp$ values. Additionally, as discussed above, we require the protocol to satisfy the *binding* property: that is, at the time t at which the first non-faulty party terminates, there is a value $v \in \{0, 1\}$ such that no non-faulty party can output v in any extension of the execution. Inspired by Gradecast [18], we also extend BCA to *Graded-BCA*, where each party outputs a value and a grade $g \in \{0, 1, 2\}$, intuitively corresponding to how confident they are in their output. Roughly speaking, in Graded-BCA, a grade of 2 implies that all parties have the same value. There are numerous results that use the graded approach [5, 20, 23]; our work is one of the few that use it explicitly for an asynchronous model.

We show that asynchronous agreement can be solved with rounds of BCA followed by a strong coin flip, or rounds of Graded-BCA followed by a flip of an ϵ-good coin. We treat this as a *framework* which can solve many variants of agreement, including crash-tolerant, information-theoretic, and computational, depending on the implementation and guarantees that the plugged in (Graded) BCA provides. We then present several implementations of (Graded) BCA, and by optimizing those, arrive at algorithms for agreement that match or improve upon the state of the art algorithms in terms of expected communication rounds to termination.

We note that in this paper we consider the *worst case* expected number of rounds until *termination*. This includes rounds after parties commit a value during which they cannot yet terminate the protocol (this accounting is not always included in the literature). Furthermore, we note that optimizations for 'optimistic' runs with synchrony or in which all non-faulty parties have the same input, are useful in practice, but are not the topic of this paper.

1.1 Our contributions: Crash failures

Asynchronous agreement can be solved while tolerating a minority of crash failures using randomization. This was first showed by Ben-Or in 1983 [4], but only against an oblivious adversary. Later, Aguilera and Toueg [2] showed that Ben-Or's protocol in fact works against an adaptive adversary as well. However, Aguilera and Toueg only considered a local coin, and showed expected termination in $O(2^{2n})$. Their results do not easily generalize to better coins. Using our framework, we obtain the following results.

Theorem 1.1. *Consider the problem of Asynchronous Binary Crash-tolerant Agreement with adaptive security, optimal resilience, asymptotically optimal message complexity for a network of n parties, where $t < \frac{n}{2}$ parties may be faulty*

(1) *Given an ϵ-good t-unpredictable common coin, there exists a protocol that reaches termination in $3/\epsilon + 4$ communication rounds in expectation.*
(2) *Given a strong t-unpredictable common coin, there exists a protocol that reaches termination in 7 communication rounds in expectation.*

Compared to Aguilera and Toueg [2]: (1) for the local common coin this is a quadratic improvement of the expected termination from $O(2^{2n})$ to $O(2^n)$; (2) for any ϵ-good common coin, when $\epsilon \gg 2^{-n}$ this provides a significantly better bound (exponential improvement for a $O(1)$-good common coin); (3) For a strong common coin, this provides the first result with adaptive security that are optimized for this setting.

	Aguilera and Toueg[2]	Ours
Strong Coin	-	7
Weak Coin	-	$3/\epsilon$+4
Local Coin	$O(2^{2n})$	$O(2^n)$

Table 1. A comparison of previous results to ours in the crash-fault setting.

	[28]	[9]	[11]	Ours
Strong $t+1$	-	-	-	17
Strong $2t+1$	-	15	13	13
Weak $t+1$	$12/\epsilon+9$	-	$6/\epsilon+6$	$6/\epsilon+6$
Strong $2t+1$ w. Threshold Sigs	-	-	-	9

Table 2. A comparison of our results in the Byzantine setting against previous results by coin strength. To the best of our knowledge, [11] has not been peer reviewed.

1.2 Our contributions: Byzantine failures in the authenticated setting

In addition to our results in the crash fault setting, we present an efficient protocol for Asynchronous Binary Byzantine Agreement with threshold signatures. This result is relevant for the several papers, such as the work of Cachin, Kursawe and Shoup [8], as well as HoneyBadger [26], Beat [17], and DUMBO-MVBA [24], that use binary agreement and assume a DKG setup, and hence may benefit from this result for adaptive security and better round complexity. Other systems [12, 13] in the authenticated model could use this result to provide liveness in asynchrony.

Theorem 1.2. *Given a strong $2t$-unpredictable coin, and a Threshold Signature Setup, there exists a protocol solving Authenticated Asynchronous Binary Byzantine Agreement with adaptive security against a computationally bounded adversary, optimal resilience, and asymptotically optimal message complexity that reaches termination in 9 communication rounds in expectation.*

1.3 Our contributions: Byzantine failures in the information theoretic setting

Ben-Or's [4] Byzantine resilient protocol does not have optimal resilience. Bracha's [6] protocol has optimal resilience but requires $O(n^3)$ message complexity, while the asymptotically optimal message complexity is $O(n^2)$ [16]. Mostefaoui, Moumen, and Reynal [27] solve Asynchronous Binary Agreement with the asymptotically optimal $O(n^2)$ message complexity and expected $O(1/\epsilon)$ rounds given an ϵ-good coin. This celebrated paper won the PODC 2014 best paper award, but it was later discovered that the protocol is not secure against adaptive adversaries. To remedy this problem, the JACM version of Mostefaoui, Moumen, and Reynal [28] had major protocol changes, more than doubling the round complexity. In particular, the worst case expected round complexity of [28] is $12/\epsilon + 9$ so even with a perfect coin the expected termination is 24+9=33 rounds. Cachin and Zanolini [9] suggest an improvement given a strong coin, but we show this improvement is not adaptively secure (liveness violation) when instantiated with a t-unpredictable strong coin.

Crain [11] in February 2020 presents improvements both for a strong coin and a weak coin. We note that these protocols, while not peer reviewed, have been used in other academic papers [14].

In this paper, we match Crain's expected rounds until termination through our framework by presenting efficient information theoretic Byzantine-tolerant implementations of (Graded-) BCA.

Theorem 1.3. *Consider the problem of Asynchronous Binary Byzantine Agreement with adaptive security, optimal resilience, asymptotically optimal message complexity for a network of n parties, where $t < \frac{n}{3}$ parties may be faulty*

(1) *Given an ϵ-good t-unpredictable common coin, there exists a protocol that reaches termination in $6/\epsilon + 6$ communication rounds in expectation.*
(2) *Given a strong t-unpredictable common coin, there exists a protocol that reaches termination in 17 communication rounds in expectation.*
(3) *Given a strong $2t$-unpredictable common coin, there exists a protocol that reaches termination in 13 communication rounds in expectation.*

We believe that our approach is simpler than that of Crain, further highlighting the advantage of our agreement framework; it allows to match state-of-the-art algorithms while abstracting out key properties, making proofs modular, and generalizing to other settings as well.

2 MODEL

We consider an asynchronous message passing system with n parties, up to t of which may fail. That is, processes may communicate with one another by sending messages on links that do not drop messages, but may take an arbitrarily long amount of time to be received (all messages do eventually reach their destination). We consider two types of failures; *crash* failures, in which a party stops executing, and *Byzantine* failures, in which a party may act arbitrarily. Any party that does not fail in either manner is called *non-faulty*. An *adaptive adversary* controls which parties fail, and the amount of delay each message experiences. The adversary does not know the future, but does have access to all parties' private state. In particular, the adversary knows the value of random coin flips at the moment that this value is revealed to the party that flipped the coin.

We consider the following abstraction for random coins.

Definition 2.1 (ϵ-Good , d-Unpredictable Coin). An *ϵ-good d-unpredictable coin* provides an *access* primitive that takes no arguments and outputs either 0 or 1 with the following properties:

- The output value for all parties is completely unpredictable until at least $d + 1$ parties have accessed the coin.
- With probability at least ϵ, all non-faulty parties get 0, and with probability at least ϵ, all non-faulty parties get 1.

An *ϵ-good and d-unpredictable* common coin has two properties: (1) *Unpredictability* the adversary cannot predict the coin value at long as at most d parties started the protocol; (2) *Good*: for any value $v \in \{0, 1\}$, all non-faulty parties output v with probability $\geq \epsilon$. If a coin is $1/2$-good, we call it a *strong coin*. Many $O(1)$-good common coins are based on Verifiable Secret Sharing [1, 7, 18]. A *strong* common coin and can be implemented in the authenticated setting using secret sharing [8, 30]. Building coins of various goodness has been studied in other works and is not the topic of this paper. Instead, we use ϵ-good coins as a black box. Since many strong common coins are based on polynomial secret sharing, we will interchangeably say d-unpredictable and degree d.

We are interested in solving binary agreement problems in this setting. In particular, we consider the following two standard variants of agreement, which correspond to a setting with Byzantine failures and a setting with only crash failures, respectively.

Definition 2.2 (Asynchronous Byzantine Agreement (ABA)). An asynchronous Byzantine agreement protocol among n parties is one in which each party receives an input and outputs a value v and has the following guarantees:

- **(Agreement)** If a non-faulty party commits v and another non-faulty party commits v', $v = v'$.
- **(Validity)** If all non-faulty parties receive the same value v as input, all non-faulty parties commit v.

- **(Termination)** All non-faulty parties eventually commit and then eventually terminate with probability 1.

*Definition 2.3 (Asynchronous Crash Agreement (*ACA*)).* A crash consensus protocol among n parties is one in which each party receives an input and outputs a value v and has the following guarantees:

- **(Agreement)** If a party commits v and another party commits v', $v = v'$.
- **(Weak Validity)** If all parties receive the same input value v, all non-faulty parties commit v.
- **(Termination)** All non-faulty parties eventually commit and then eventually terminate with probability 1.

ACA is sometimes referred to as crash-tolerant consensus, uniform consensus, or simply *consensus* in the literature. The definition for ACA differs from that of ABA in that the agreement property of ACA is stronger as it must apply for *any two parties* that output values. The validity property of ACA is weaker in that it only applies if *all parties* start the protocol with the same value. These differences in definition are standard and reflect the nature of failures assumed for these two problems; generally, it is assumed that parties executing ACA only fail by crashing, whereas parties executing ABA can fail arbitrarily. The weaker validity property is needed to allow ACA to be solvable with $2f + 1$ parties. In this paper, when referring generally to either one of these agreement problems, we say *Asynchronous Agreement (*AA*)*. We note that in this paper, we only consider Asynchronous Agreement with binary input. We therefore drop the 'binary' qualifier in the rest of the paper.

2.1 Notations

For the reader's convenience, we provide here all the notations we use to refer to different problem variants throughout this paper, including ones defined later on. The notations are also introduced in the first place that uses them.

We use the acronym $ABA_{\frac{1}{2}}$ to refer to Asynchronous Byzantine Agreement with a strong coin. ABA_{ϵ} refers to Asynchronous Byzantine Agreement with an ϵ-good coin. BCA_{Byz} refers to Binding Crusader Agreement for a Byzantine adversary, while BCA_{Crash} refers to Binding Crusader Agreement for a crash adversary. $BCA_{Byz,TSig}$ refers to Binding Crusader Agreement with threshold signatures for a Byzantine adversary. $EVBCA_{Byz}$, $EVGBCA_{Byz}$, and $EVBCA_{Byz,TSig}$ refer to the Externally Valid BCA, Externally Valid Graded BCA, and Externally Valid BCA with threshold signatures protocols in the Byzantine setting. The notation $ABA_{\frac{1}{2}}$-$BCA_{Byz,TSig}$ refers to the $ABA_{\frac{1}{2}}$ protocol with $BCA_{Byz,TSig}$ plugged in for BCA.

3 A FRAMEWORK FOR RANDOMIZED AGREEMENT

In Appendix A, we describe Asynchronous Byzantine Agreement algorithms of the past that had flawed proofs of termination. They all follow a common pattern; parties execute rounds in which they try to decide a value, and if they cannot decide, they flip a coin to use as their input value in the next round. The counter examples to these algorithms similarly all stem from the same core problem: after the coin value is revealed, the adversary can still influence some parties to choose either value, and in particular, can force some of them not to adopt the coin value for the next round.

Our main contribution in this paper is capturing this common algorithmic pattern in a simple unifying framework, which proceeds in rounds of *crusader agreement* followed by a coin flip. Crusader agreement [15] allows parties to decide $\perp$ instead of one of the input values if not all parties received the same input. Intuitively, in our framework a party adopts the value of the coin in round r if and only if the decision of its rth crusader agreement instance was $\perp$.

To make this framework solve asynchronous agreement against an adaptive adversary, we introduce a new property, called *binding*, to crusader agreement. The binding property dictates that by the time at which the first

non-faulty party decides in an instance of crusader agreement, there is some value that no non-faulty party can decide. Thus, in the context of our agreement framework, this means that the adversary must 'bind' to some value before it sees the value of the coin in each round. Binding Crusader Agreement (BCA), is thus defined as follows.

Definition 3.1 (Binding Crusader Agreement (BCA)). A Binding Crusader Agreement protocol between n parties has the following guarantees:

> **(Agreement)** If two non-faulty parties decide values x and y, then either $x = y$ or at least one of the values is $\perp$.
> **(Validity)** If all non-faulty parties have the same input, then this is the only possible decision for non-faulty parties.
> **(Termination)** All non-faulty parties eventually decide and then eventually terminate.
> **(Binding)** Let time τ be the first time such that there is a party that is non-faulty at time τ and decides at time τ. At time τ, there is a value $b \in \{0, 1\}$ such that no non-faulty party decides $1 - b$ in any extension of this execution.

Note that the binding property is only interesting in the case that the non-faulty party referred to in the definition decided $\perp$. Otherwise, it trivially follows from agreement. To be able to implement BCA with an optimal tolerance to crash faults, we must weaken its validity property to match that of ACA. In particular, we must use the following validity property:

- **(Weak Validity)** If all parties have the same input v, then all non-faulty parties decide v.

We refer to BCA with this weak validity property as BCA-Crash, and to the one with the stronger validity property as BCA-Byz. That is, if any party, even a faulty one, has input value v to an instance of BCA-Crash, the protocol is allowed to decide v. This is known as a *weak validity* property, and has been shown to be necessary for solving agreement in an asynchronous setting with $n \geq 2t + 1$ parties, even under crash faults [3, 25]. The proof of that result can easily be modified to apply to BCA as well.

The full definition of BCA-Crash appears in Appendix B. Other changes include requiring that *all* parties that terminate satisfy agreement (not just the non-faulty ones). Similarly, binding applies to all parties, including faulty ones.

Any solution to BCA (of either kind) can then be plugged into our framework to yield a solution to asynchronous agreement. The properties of the BCA implementation directly translate to properties of the resulting agreement protocol. For example, the agreement protocol inherits its fault tolerance and validity from its BCA. Thus, the same framework captures both crash-tolerant and Byzantine-tolerant agreement.

We present two versions of the framework depending on the strength of the coin that is used. The first uses BCA as was explained above, but requires a strong coin. That is, in each round, the coin must give the same value to all parties. To handle a weaker coin guarantee, we replace binding crusader agreement with its *graded* version. Graded crusader agreement was introduced by Feldman and Micali [18], and has parties output, along with their decision value, a *grade* that intuitively indicates how sure they are of their decision value. The definition of GBCA is as follows.

Definition 3.2 (Graded Binding Crusader Agreement (GBCA)). In a Graded Binding Crusader Agreement protocol, each party decides a value from a set of 5 ordered buckets: 0 grade 2, 0 grade 1, $\perp$ grade 0, 1 grade 1, 1 grade 2 and has the following guarantees:

> **(Graded Agreement)** If a non-faulty party decides v grade 2 or v grade 1, no non-faulty party decides $1 - v$ grade 1 or 2. Further, if a non-faulty party decides v grade 2, no non-faulty party decides $\perp$ grade 0.
> **(Validity)** If all non-faulty parties have the same input v, then all non-faulty parties decide v grade 2.
> **(Termination)** All non-faulty parties eventually decide and then eventually terminate.

(Graded Binding) Let time τ be the first time such that there is a party that is non-faulty at time τ and decides at time τ. At time τ, there is a value $b \in \{0, 1\}$ such that no non-faulty party decides $1 - b$ grade 1 or 2 in any extension of this execution.

We call this version of GBCA GBCA-Byz, and define GBCA-Crash by weakening the validity property as with the non-graded version. The full definition of GBCA-Crash appears in Appendix B.

With graded BCA, parties in our framework only commit in their agreement instance if the decision of their GBCA was grade 2 (high confidence), and only adopt the coin if their decision was grade 0 (equivalently, if they decide $\perp$). If a party decides a non-$\perp$ value v with grade 1 in round r of the framework, it will adopt v as the input for the next round, but will not decide yet. Of course, this version of the protocol works with a strong coin as well, but as we show later on, BCA can be solved more efficiently than its graded counterpart, and therefore our strong-coin version of the framework allows for more efficient solutions.

In the rest of this section, we present pseudocode for the two versions of our framework and prove that the Byzantine and Crash tolerant versions of (graded) BCA can be pluggged in to yield solutions to ABA and ACA respectively.

A note on termination. In our framework, when a party commits a value for the agreement protocol, it continues participating until all others commit as well. Achieving an actual termination point can be done by having every non-faulty party broadcasting a special "commited" message upon deciding a value v. In the crash setting, a party can commit v once it receives such a message, broadcast the message, and terminate. In the Byzantine setting, once a party receives $t + 1$ such messages, they can commit v and broadcast the same message. Upon receiving $2t + 1$ such messages, a party can terminate. Therefore, in this section we simply measure the number of high level BCA-coin rounds until all parties commit. In later sections, we measure *broadcasts*, i.e., the number of communication steps required to terminate in an implementation of BCA. There, we report one extra broadcast to account for the commitment broadcast. Thus, in all our theorems we report rounds until actual termination (not just until commitment).

3.1 Asynchronous Agreement with a Strong Coin

Algorithm 1: Asynchronous Agreement with a Strong Coin ($AA_{\frac{1}{2}}$)

Input: x

1: $r = 0$, $est = x$;
2: **while** *true*
3: $r+ = 1$
4: $val = \mathsf{BCA}(est)$
5: $c = \mathsf{CommonCoin}()$
6: **if** $val \neq \perp$ and $c == val$ **then** commit val, $est = val$
7: **else if** $val \neq \perp$ **then** $est = val$
8: **else** $est = c$;

Our framework for solving asynchronous agreement with a strong coin is presented in Algorithm 1. As described above, it proceeds in rounds of BCA followed by a coin flip. A party that decides $\perp$ in its BCA instance of round r adopts the coin value of r as the input to the BCA instance in round $r + 1$. A party that decides a non-$\perp$ value v in round r will not adopt the coin value. Instead, if the coin value is also v, then it commits v. Otherwise, it uses v as its input to the BCA of round $r + 1$.

Intuitively, a party that commits a value v in round r knows that it is safe to do so since all other parties will either have decide v in r as well, and therefore will have committed in round r as well, or they will have seen that the value of the coin is v, and will therefore adopt it in the next round. All non-faulty parties are therefore guaranteed to execute the next round of BCA with input v, and are therefore guaranteed to decide that value until they eventually commit it.

In the rest of this subsection, we show that by plugging in a Byzantine-tolerant version of BCA into Algorithm 1, we get a solution to ABA, and by plugging in a crash-tolerant BCA implementation, we get a solution to ACA. We note that ACA has a weaker validity property than ABA, and accordingly we define a weak validity version of BCA to allow for more fault tolerant implementations in the case of crash faults.

We now prove that this framework is correct. Theorem 3.3 formalizes this statement.

Theorem 3.3. *If we plug an implementation of* BCA-Byz *into* BCA*, and* CommonCoin *is a correct implementation of a strong coin, then Algorithm 1 solves Asynchronous Byzantine Agreement in any system in which* $n \geq 3t + 1$ *and terminates in 4 rounds in expectation against an adaptive adversary.*

To prove Theorem 3.3, we start with a useful lemma.

Lemma 3.4. *If all non-faulty parties start a round* r *with the same estimate value* $est = v$*, then all non-faulty parties will commit* v *within a constant number of rounds.*

Proof. By the validity of BCA, since every non-faulty party inputs v to the instance of BCA, all non-faulty parties decide v. If $c = v$, all non-faulty commit v in this round. Otherwise, all non-faulty parties set $est = v$ and start the next round. This continues for each round until we reach a round in which $c = v$, which happens in an expected constant number of rounds from round r. □

Using Lemma 3.4, it is now easy to show the correctness of the $AA_{\frac{1}{2}}$ protocol (Algorithm 1).

Proof of Theorem 3.3. We prove that each of the properties of asynchronous Byzantine agreement is satisfied.

Agreement. Note that a non-faulty party p only commits a value in a round r if the coin agreed with this value and it was the decision of p's BCA in round r. Let p be the first non-faulty party that commits, let r be the round in which it commits, and let v be its committed value. Note that in round r, no other value can be commited, since the commitment value is always the same as the coin of that round, and the coin is strong, so for any round r, all parties have the same coin value in r. Recall that BCA ensures that if a non-faulty party decides v from BCA, every other non-faulty party decides either $\perp$ or v from that instance of BCA. Since the coin value was v in r, all non-faulty parties must have started round $r + 1$ with $est = v$. Therefore, by Lemma 3.4, every other correct party q must also decide v.

Validity. If all non-faulty parties have the same input, then they all start round 0 with the same estimate. Therefore, by Lemma 3.4, they will all decide on that input.

Termination. Note that if in any round, either all non-faulty parties decide $\perp$ in their BCA, or the coin is the same as the non-$\perp$ decision value of BCA, then the lemma applies in the next round. Furthermore, by the binding property of BCA, the adversary is bound to the non-$\perp$ decision value of the BCA in any round r (if there is one) by the time the first non-faulty party finishes its BCA in round r. In particular, this must happen before the coin value is revealed in any coin of degree t or larger. Therefore, in each round, there is at least a 50% chance that the BCA decision value is the same as the coin, or $\perp$. □

We now show that the framework presented in Algorithm 1 also works for the crash tolerant case with BCA-Crash. Theorem 3.3 almost immediately implies that the frameworks work for the crash-fault setting as well, but we must ensure that the weaker validity property of BCA-Crash does not break correctness.

Theorem 3.5. *If we plug an implementation of* BCA-Crash *into* BCA*, and* CommonCoin *is a correct implementation of a strong coin, then Algorithm 1 solves Asynchronous Crash Agreement in any system in which* $n \geq 2t + 1$ *and terminates in 4 rounds in expectation against an adaptive adversary.*

The proof of Theorem 3.5 can be found in Appendix C.1.

3.2 Asynchronous Agreement for Weak Coin from Graded BCA

The framework presented in Algorithm 1 relies heavily on the common coin giving the same random value to each non-faulty party in a given round. If the coin were to sometimes give different values to different parties, Lemma 3.4 would no longer hold, which could lead to agreement breaking.

More concretely, the following bad scenario could occur: a non-faulty party x decides value v in its BCA in a round r, and gets v from its coin in round r as well, and therefore commits v. However, all other non-faulty processes decide $\perp$ in their BCA in round r and receive a coin value of $1 - v$. In this case, it is possible that these non-faulty parties decide $1 - v$ in the BCA of round $r + 1$, and could potentially commit $1 - v$ if their coin for round $r + 1$ agrees with this value. In this way, agreement would be violated.

However, implementing a strong common coin requires cryptographic support [8]. Thus, it is important to ask whether we can solve AA using a similar framework, while only making use of a *weak coin*.

Algorithm 2: Asynchronous Agreement with a Weak Coin (AA_ϵ)

Input: x

1: $v = x$
2: **while** true
3: $\quad val, grade = \texttt{GBCA}(v)$
4: $\quad c = \texttt{WeakCoin}()$
5: $\quad$ **if** $val == \perp$ **then** $v = c$
6: $\quad$ **else** $v = val$
7: $\quad$ **if** $grade = 2$ **then** commit val

In this subsection, we show that this is possible by tweaking the BCA abstraction. In particular, as described in the beginning of Section 3, we make use of a gradecast, or *graded crusader agreement* abstraction, as introduced by Feldman and Micali [18].

We now show how GBCA can be used to solve AA using a weak coin. Pseudocode for this algorithm is presented in Algorithm 2. The idea is simple; the algorithm proceeds in rounds in which GBCA is run, followed by a coin flip. Intuitively, the coin and the decision of this round's GBCA together inform the input for the next round's GBCA. However, the coin is only ever adopted if the decision of GBCA was $\perp$. Otherwise, if the decision was some non-$\perp$ value v, regardless of the grade, v will be this party's input for the next round. Furthermore, if the decision was grade 2, then the party commits v in this round. The key insight is that if a party decides grade 2 v and commits, then by graded agreement all others will have v as input for their next round, and will therefore all commit in the next round by validity. We formalize this argument by proving that plugging in implementations of GBCA-Byz and GBCA-Crash into Algorithm 2 yields solutions to ABA and ACA respectively regardless of the strength of the coin. The proofs of the following theorems for can be found in Appendix C.2.

Theorem 3.6. *If we plug an implementation of* GBCA-Byz *into* GBCA*, and* WeakCoin *is a correct implementation of an* ϵ*-good coin, then Algorithm 2 solves Asynchronous Byzantine Agreement in any system in which* $n \geq 3t + 1$ *and terminates in* $1 + 1/\epsilon$ *expected rounds against an adaptive adversary.*

Theorem 3.7. *If we plug an implementation of* GBCA-Crash *into* GBCA*, and* WeakCoin *is a correct implementation of an ϵ-good coin, then Algorithm 2 solves Asynchronous Crash Agreement in any system in which $n \geq 2t+1$ and terminates in $1 + 1/\epsilon$ expected rounds against an adaptive adversary.*

4 IMPLEMENTING BINDING CRUSADER AGREEMENT

4.1 Asynchronous Crash Fault Tolerant Binding Crusader Agreement

We now present an algorithm for weak validity BCA tolerating $t < n/2$ crash faults. The pseudocode is shown in Algorithm 3. To participate in the algorithm, a party p begins by sending its input value x to all parties in the system, labeled as a 'value message' with the tag `val`. It then waits to receive at least $n - t$ value messages, and checks whether they all contain the same value x. If so, p sends an echo message $\langle \text{echo}, id, x \rangle$ to all, and otherwise, it echos $\perp$, to indicate that it has witnessed at least two different initial values. p then waits to receive at least $n - t$ echo messages; if all of them again contain the same value x, p outputs x as its decision value. Otherwise, p outputs $\perp$.

Algorithm 3: Asynchronous Binding Crusader Agreement for Crash Faults instance $id(BCA_{Crash})$

Input: x

1: send $\langle \text{val}, id, x \rangle$ to all
2: **upon** receiving $\langle \text{val}, id, * \rangle$ messages from $n - t$ parties and not having sent an echo message:
3: **if** all of the messages contain the same value x, send $\langle \text{echo}, id, x \rangle$ to all
4: **else**, send $\langle \text{echo}, id, \perp \rangle$ to all
5: **upon** receiving $\langle \text{echo}, id, * \rangle$ messages from $n - t$ parties:
6: **if** all the messages contain the same value x, decide x
7: **else**, decide $\perp$

We the proof of correctness for Algorithm 3 showing it is an implementation of weak validity BCA for crash faults with optimal fault tolerance appears in Appendix D.1. Here we simply state the main theorems that yield the result.

Theorem 4.1. *Algorithm 3 implements weak validity* BCA *(*BCA-Crash*) tolerating $t < \lceil n/2 \rceil$ crash faults and all parties decide within 2 communication rounds.*

Theorem 4.1 and Theorem 3.5 together imply the following result.

Theorem 4.2. *There is an algorithm that solves asynchronous crash-tolerant agreement in an expected 7 broadcasts in a system with $n \geq 2t+1$ parties and a t-unpredictable strong common coin.*

4.2 Asynchronous Byzantine Fault Tolerant Binding Crusader Agreement

We now present an implementation of BCA that tolerates Byzantine failures, and works as long as $n \geq 3t+1$. The algorithm is presented in Algorithm 4. In the pseudocode, we use **upon** to specify a clause that is triggered every time the specified condition is met, and **wait until** to indicate a clause that is only triggered *the first time* its condition is met.

The algorithm proceeds as follows. Like in the crash-tolerant case, a party starts by sending its input value x to all other parties. In contrast to the crash-tolerant case though, here we have parties *amplify* values that they hear from at least one non-faulty party (i.e., $t+1$ parties in total). The initial send of the value and the amplification are treated in the same way, and thus we call both types of messages an *echo* message.

Once $\langle \text{echo}, v \rangle$ for the same v is received from at least $n - t$ parties, a party p sends an *echo2* message with value v, and adds v to its $approvedVals$. Intuitively, p will never propagate or decide a value that isn't in its $approvedVals$ set. The $approvedVals$ set can later be updated to contain the other value $1 - v$ as well if $n - t$ $\langle \text{echo}, id, 1 - v \rangle$ messages are received. However, a non-faulty party only ever sends one echo2 message.

Intuitively, since each non-faulty party sends just one echo2 message, and that message always contains a non-$\perp$ value, we can think of echo2 messages as 'voting' for a value that we are sure all non-faulty parties will eventually hear and that some non-faulty party received as input. In the next phase of the protocol, a non-faulty party p aggregates the received votes and either amplifies a value v by sending $\langle \text{echo3}, id, v \rangle$ if it heard $n - t$ distinct parties vote for v with an $\langle \text{echo2}, id, v \rangle$ message, or sends $\langle \text{echo3}, id, \perp \rangle$ if it heard differing votes. An $\langle \text{echo3}, id, \perp \rangle$ message can also be sent if p has placed both v and $1 - v$ in its $approvedVals$ set, meaning that eventually all parties will see and have both values in their $approvedVals$ sets as well.

Finally, p makes its decision by aggregating echo3 messages. If it receives at least $n - t$ echo3 messages with the same non-$\perp$ value v, it decides v. Otherwise, it waits until both v and $1 - v$ are in its $approvedVals$ and it has received $n - t$ echo3 messages (regardless of their value) and then decides $\perp$. Intuitively, waiting to have both values in the $approvedVals$ helps validity; it ensures that the Byzantine parties can't force a decision of $\perp$ if all non-faulty parties started with the same value.

Algorithm 4: Asynchronous Binding Crusader Agreement for Byzantine Faults instance id (BCA_{Byz})

Input: x

$approvedVals = \{\}$
send $\langle \text{echo}, id, x \rangle$ to all
upon receiving $\langle \text{echo}, id, v \rangle$ for the same non-$\perp$ v from $t + 1$ parties:
  if haven't echoed v yet **then** send $\langle \text{echo}, id, v \rangle$ to all
upon receiving $\langle \text{echo}, id, v \rangle$ for the same non-$\perp$ v from $n - t$ parties:
  $approvedVals = approvedVals \bigcup v$
  if haven't sent any echo2 message yet **then** send $\langle \text{echo2}, id, v \rangle$ to all
wait until
    (1) $|approvedVals| > 1$ or
    (2) received $\langle \text{echo2}, id, v \rangle$ for the same non-$\perp$ v from $n - t$ parties
  if (1) **then** send $\langle \text{echo3}, id, \perp \rangle$ to all
  else send $\langle \text{echo3}, id, v \rangle$ to all
wait until
    (1) $|approvedVals| > 1$ and received echo3 messages from at least $n - t$ parties or
    (2) received $\langle \text{echo3}, id, v \rangle$ for the same non-$\perp$ v from at least $n - t$ parties
  if (1) **then** decide $\perp$
  else decide v

We now prove that Algorithm 4 satisfies the properties stated in Definition 3.1. More specifically, we show the following theorem, which is implied from Lemmas 4.4, 4.5, 4.7 and 4.9.

Theorem 4.3. *Algorithm 4 implements binding crusader agreement that terminates in at most 4 communication rounds and works in an asynchronous system with $n \geq 3t + 1$ parties where t parties could be Byzantine.*

Lemma 4.4. *Algorithm 4 satisfies agreement.*

Proof. Assume, for a contradiction, that two non-faulty parties p_i and p_j decide non-$\perp$ values x_i and x_j respectively, such that $x_i \neq x_j$. In order to decide x_i, p_i must have received echo3 messages from $n - t$ different

parties with the value x_i. Similarly, p_j must have received echo3 messages from $n - t$ different parties with the value x_j. By quorum intersection this cannot happen since each non-faulty party sends only one echo3 message. □

Lemma 4.5. *Algorithm 4 satisfies validity.*

Proof. We prove validity in two parts. First, we prove that if all non-faulty parties start with value v, no non-faulty party outputs $\perp$. Assume, for a contradiction, that all non-faulty parties start with v and some non-faulty party p_i outputs $\perp$. Notice that the condition under which a party decides $\perp$ (line 16) requires that that party's $approvedVals$ set contains two distinct values. In order for a party to have two values in its $approvedVals$ set, it must receive $n - t$ copies of $\langle \text{echo}, b \rangle$ for both $b = 0$ and $b = 1$. A party only sends an amplification echo for a value $v' \neq v$ (line 4) if they received at least $t + 1$ messages $\langle \text{echo}, id, v' \rangle$. Therefore, some non-faulty party must have started with value $1 - v$, a contradiction.

Next, we prove that if all non-faulty parties start with value v, no non-faulty party decides $1 - v$. In order to decide $1 - v$, a non-faulty party must have $1 - v$ in its $approvedVals$ set (line 15). Again, this requires that some non-faulty party started with $1 - v$, a contradiction. □

Lemma 4.6. *If all non-faulty parties begin the protocol in Algorithm 4 and no non-faulty party terminates, the $approvedVals$ sets of all non-faulty parties are eventually equal.*

Proof. A non-faulty party adds value v to its $approvedVals$ set after receiving $\langle \text{echo}, id, v \rangle$ messages from at least $n - t$ distinct parties, at least $n - 2t$ of which had to have been non-faulty parties. Therefore, if a non-faulty party adds v to its $approvedVals$ set, all non-faulty parties receive at least $t + 1$ $\langle \text{echo}, id, v \rangle$ messages and send $\langle \text{echo}, id, v \rangle$ themselves if they haven't already (line 4). As a result, all non-faulty parties receive $\langle \text{echo}, id, v \rangle$ messages from at least $n - t$ parties and add v to their $approvedVals$ sets. □

Lemma 4.7. *Algorithm 4 satisfies termination within 4 communication rounds.*

Proof. Since we are in the binary agreement case, it is clear by the pseudocode that a party sends at most 4 messages; either one or two echo messages, one echo2 message and one echo3 message.

Also because we are considering the binary agreement case, there must be some value v such that at least $t + 1$ non-faulty parties start with v. Therefore, eventually all non-faulty parties send $\langle \text{echo}, id, v \rangle$. All non-faulty parties will then receive at least $n - t$ $\langle \text{echo}, id, v \rangle$ messages and send an echo2 message if they haven't already. In addition, each party will add v to its $approvedVals$ set.

In order for a non-faulty party to proceed in the protocol by sending an echo3 message, either the condition on Line 10 or Line 9 must be reached and satisfied. Note that non-faulty parties only send echo2 messages containing non-$\perp$ values. We consider two cases to prove that every non-faulty party eventually sends an echo3 message:

(1) All non-faulty parties send echo2 messages for the same value v. Thus the condition on Line 10 is triggered and met once it receives echo2 messages from all non-faulty parties.
(2) Some non-faulty party sends $\langle \text{echo2}, id, v \rangle$, and some non-faulty party sends $\langle \text{echo2}, id, 1 - v \rangle$. By Lemma 4.6, the $approvedVals$ sets of all non-faulty parties eventually contain both v and $1 - v$. Therefore, eventually the condition on Line 9 will be met.

We have thus proved that all non-faulty parties send echo3 messages, and we must now prove that all non-faulty parties decide. We therefore must show that either the condition on Line 14 or the condition on Line 15 is reached and satisfied for all non-faulty parties that have not yet decided. We consider two cases:

(1) All non-faulty parties send $\langle \text{echo3}, id, v \rangle$ for the same non-$\perp$ value v. So clearly, Line 15 will be reached and satisfied.

(2) Not all non-faulty parties send $\langle \text{echo3}, id, v \rangle$ for the same non-$\perp$ value v. If a non-faulty party sent $\langle \text{echo3}, id, \perp \rangle$, then its $approvedVals$ must have had more than one value in it. Furthermore, since any non-faulty party that sends $\langle \text{echo2}, id, v \rangle$ must have v in its $approvedVals$, and a non-faulty party must receive $\langle \text{echo2}, id, v \rangle$ from at least $t + 1$ non-faulty parties before sending $\langle \text{echo3}, id, v \rangle$, any non-$\perp$ value that was sent in an echo3 message by a non-faulty party must be in some non-faulty party's $approvedVals$. By Lemma 4.6, eventually the $approvedVals$ sets of all non-faulty parties contain both 0 and 1. Since we've shown above that all non-faulty parties eventually send an echo3 message, the condition on Line 14 will eventually be satisfied.

Therefore, all non-faulty parties eventually decide. □

Lemma 4.8. *If a non-faulty party sends* $\langle \text{echo3}, id, v \rangle$*, where* v *is a non-bot value, no non-faulty party sends* $\langle \text{echo3}, id, 1 - v \rangle$.

Proof. If a non-faulty party sends $\langle \text{echo3}, id, v \rangle$ for a non-$\perp$ v they received $\langle \text{echo2}, id, v \rangle$ from at least $n - t$ parties. Since non-faulty parties send a single echo2 message, no non-faulty party can receive a sufficient number of $\langle \text{echo2}, id, 1 - v \rangle$ messages to send $\langle \text{echo3}, id, 1 - v \rangle$. □

Lemma 4.9. *Algorithm 4 satisfies binding.*

Proof. Let p_i be the first non-faulty party to decide. p_i must have received at least $n - t$ echo3 messages, at least $t + 1$ of which were from non-faulty parties. Consider these $t + 1$ echo3 messages from non-faulty parties. If one of these messages contained a non-$\perp$ value v, then by Lemma 4.8, no non-faulty party sends $\langle \text{echo3}, id, 1 - v \rangle$ and therefore no non-faulty party can decide $1 - v$. If all of the $t + 1$ echo3 messages contain $\perp$, then no non-faulty party can receive enough echo3 messages with either non-$\perp$ values to decide either non-$\perp$ value. □

In Appendix G.1 we show how, when plugging BCA_{Byz} into $AA_{\frac{1}{2}}$, we can use information from the $AA_{\frac{1}{2}}$ execution, such as the value of the coin in the previous round, to reduce the number of broadcasts per round. We define a new primitive based on BCA titled Externally Valid Binding Crusader Agreement (EVBCA) and present $EVBCA_{Byz}$, a modified version of the BCA_{Byz} protocol that implements EVBCA.

Theorem 4.10. *There is an algorithm that solves asynchronous Byzantine fault tolerant agreement in expected 13 broadcasts in a system with* $n \geq 3t + 1$ *parties and a* $2t$*-unpredictable strong coin.*

Theorem 4.11. *There is an algorithm that solves asynchronous Byzantine fault tolerant agreement in an expected 17 broadcasts in a system with* $n \geq 3t + 1$ *parties and a* t*-unpredictable strong coin.*

5 IMPLEMENTING GRADED BCA

5.1 Asynchronous Crash Fault Tolerant GBCA

In Algorithm 5, we present a protocol for Asynchronous Graded Binding Crusader Agreement that withstands $n \geq 2t + 1$ crash faults. A party participating in this protocol first broadcasts their input, x, to all of the other in a val message. Upon receiving val messages from $n - t$ parties, a party broadcasts an $\langle \text{echo}, v \rangle$ message if they all contain the same value v and $\langle \text{echo}, \perp \rangle$ otherwise. Similarly, upon receiving echo messages from $n - t$ parties, a party broadcasts an $\langle \text{echo2}, v \rangle$ message if they all contain the same value v and $\langle \text{echo2}, \perp \rangle$ otherwise. Finally, once a party has received echo2 messages from $n - t$ parties, if they all contain the same non-$\perp$ value v or they all contain $\perp$, the party decides v (v grade 2) or $\perp$ respectively. If at least one of the echo2 messages contains non-$\perp$ value v and at least one of the echo2 messages contains contains a value other than v, the party decides $v\perp$ (v grade 1). Notice that up until line 8, the protocol is nearly the same as that of BCA_{Crash} in Algorithm 3. The echo2 message a party sends corresponds to the value decided in Algorithm 3. The proofs for the following theorems can be found in Appendix E.1.

Algorithm 5: Asynchronous Graded Binding Crusader Agreement for Crash Faults instance id ($GBCA_{Crash}$)

Input: x

1: send $\langle \text{val}, id, x \rangle$ to all
2: **upon** receiving $\langle \text{val}, id, * \rangle$ messages from $n - t$ parties and not having sent an echo message:
3: **if** all of the messages contain the same value v **then** send $\langle \text{echo}, id, v \rangle$ to all
4: **else** send $\langle \text{echo}, id, \perp \rangle$ to all
5: **upon** receiving $\langle \text{echo}, id, * \rangle$ messages from $n - t$ parties and not having sent an echo2 message:
6: **if** all of the messages contain the same value v **then** send $\langle \text{echo2}, id, v \rangle$ to all
7: **else** send $\langle \text{echo2}, id, \perp \rangle$ to all
8: **upon** receiving $\langle \text{echo2}, id, * \rangle$ messages from $n - t$ parties:
9: **if** all of the messages contain the same value v **then** decide v, 2
10: **else if** ≥ 1 of the messages contains v and ≥ 1 of the messages contains a value other than v **then** decide v, 1
11: **else** decide $\perp$, 0

Theorem 5.1. *Algorithm 5 implements graded binding crusader agreement that terminates in at most 3 communication rounds and works in an asynchronous system with $n \geq 2t + 1$ parties where t parties could crash.*

Theorem 5.2. *There is an algorithm that solves asynchronous crash fault tolerant agreement in expected $3/\epsilon + 4$ broadcasts in a system with $n \geq 2t + 1$ parties and an ϵ-good coin.*

5.2 Asynchronous Byzantine Fault Tolerant Graded Binding Crusader Agreement

In this section, we present a protocol for Asynchronous Graded Binding Crusader Agreement that tolerates $t < \frac{n}{3}$ Byzantine faults. This protocol can be thought of as performing BCA_{Byz} in order to bind the adversary to a single non-$\perp$ value v that no non-faulty party can output, or to having $\perp$ be the only value that non-faulty parties can output. This gives us the graded binding property of Definition 3.2. Using the output of this first BCA_{Byz}, we then add an additional echo4 and echo5 round to get the graded agreement agreement property necessary for ABA with a weak coin. Graded agreement states that if some non-faulty party outputs v grade 2, all other non-faulty parties output v with grade 2 or 1. This ensures that if some non-faulty party decides v in the ABA protocol in a given round, all non-faulty parties start the next round with v and, due to the validity property, decide v in the next round of ABA.

As such, Algorithm 6 is nearly identical to Algorithm 4 up through line 17. The echo4 message that a party sends corresponds to the value output from Algorithm 4. Upon receiving echo4 messages from $2t + 1$ parties, a party sends an echo5 message for value v if they all contain the same non-$\perp$ value v or an echo5 message for $\perp$ if its $approvedVals$ set contains more than 1 value. Finally, a party decides based on the following criteria:

(1) v grade 2 if it receives echo5 messages from $n - t$ parties for the same non-$\perp$ value v
(2) v grade 1 if it receives echo5 messages from $n - t$ parties (at least one of which contains a non-$\perp$ value v), it receives $t + 1$ echo4 messages for v, and its $approvedVals$ set contains two values
(3) $\perp$ grade 0 if it receives $n - t$ echo5 messages containing $\perp$ and its $approvedVals$ set contains two values.

By quorum intersection, it can't be the case that condition 1 is satisfied for one non-faulty party and condition 3 is satisfied for another non-faulty party in the same instance, ensuring graded agreement. In condition 2, checking that $t + 1$ parties have sent $\langle \text{echo4}, id, v \rangle$ messages before outputting v grade 1 ensures that some non-faulty party has sent $\langle \text{echo4}, id, v \rangle$ so that the binding property of Algorithm 1 holds. By checking that $approvedVals$

Algorithm 6: Asynchronous Graded Binding Crusader Agreement for Byzantine Faults instance id ($GBCA_{Byz}$)

Input: x

$approvedVals = \{\}$
send $\langle \text{echo}, id, x \rangle$ to all
upon receiving $\langle \text{echo}, id, v \rangle$ for the same non-$\perp$ v from $t+1$ parties:
  if haven't echoed v yet **then** send $\langle \text{echo}, id, v \rangle$ to all
upon receiving $\langle \text{echo}, id, v \rangle$ for the same non-$\perp$ v from $n-t$ parties:
  $approvedVals = approvedVals \bigcup v$
  if haven't sent an echo2 message yet **then** send $\langle \text{echo2}, id, v \rangle$ to all
wait until
    (1) $\langle \text{echo2}, id, v \rangle$ has been received from $n-t$ parties for the same non-$\perp$ v **or**
    (2) $\langle \text{echo2}, id, * \rangle$ messages have been received from $n-t$ parties and $|approvedVals| > 1$
  if 1 **then** send $\langle \text{echo3}, id, v \rangle$ to all
  else send $\langle \text{echo3}, id, \perp \rangle$ to all
wait until
    (1) $\langle \text{echo3}, id, v \rangle$ has been received from $n-t$ parties for the same non-$\perp$ v **or**
    (2) $\langle \text{echo3}, id, * \rangle$ messages have been received from $n-t$ parties and $|approvedVals| > 1$
  if (1) **then** send $\langle \text{echo4}, id, v \rangle$ to all
  else send $\langle \text{echo4}, id, \perp \rangle$ to all
wait until
    (1) $\langle \text{echo4}, id, v \rangle$ has been received from $n-t$ parties for the same non-$\perp$ v **or**
    (2) $\langle \text{echo4}, id, * \rangle$ messages have been received from $n-t$ distinct parties and $|approvedVals| > 1$
  if (1) **then** send $\langle \text{echo5}, id, v \rangle$ to all
  else send $\langle \text{echo5}, id, \perp \rangle$ to all
wait until
    (1) $\langle \text{echo5}, id, v \rangle$ has been received from $n-t$ parties for the same non-$\perp$ v **or**
    (2) $\langle \text{echo5}, id, * \rangle$ messages have been received from at least $n-t$ distinct parties, at least one of which is $\langle \text{echo5}, id, v \rangle$ s.t. v is non-$\perp$, $\langle \text{echo4}, id, v \rangle$ messages have been received from at least $t+1$ distinct parties, and $|approvedVals| > 1$ **or**
    (3) $\langle \text{echo5}, id, \perp \rangle$ messages have been received from at least $n-t$ distinct parties and $|approvedVals| > 1$
  if (1) **then** decide v, 2
  else if (2) **then** decide v, 1
  else decide $\perp$, 0

contains more than 1 value (conditions 2 and 3) prior to outputting v grade 1 or $\perp$ grade 0, we get validity. We prove the following results in Appendix E.2.

Theorem 5.3. *Algorithm 6 implements graded binding crusader agreement that terminates in at most 6 communication rounds and works in an asynchronous system with $n \geq 3t+1$ parties where t parties could be Byzantine.*

Theorem 5.4. *There is an algorithm that solves asynchronous Byzantine fault tolerant agreement in expected $6/\epsilon + 6$ broadcasts in a system with $n \geq 3t+1$ parties, where t parties could be Byzantine, and an ϵ-good coin.*

6 BINDING CRUSADER AGREEMENT WITH THRESHOLD SIGNATURES

In this section, we present $BCA_{Byz,TSig}$, the BCA protocol that is used in $ABA_{\frac{1}{2}}$ to implement $ABA_{\frac{1}{2}}$-$BCA_{Byz,TSig}$. $BCA_{Byz,TSig}$ tolerates $t < \frac{n}{3}$ Byzantine parties. We add two cryptographic assumptions in this model: digital signatures and threshold signatures. More specifically, we use a k out of l threshold signature scheme [30] i.e., k parties must participate in order to create a valid threshold signature. We use threshold signatures with two thresholds: for $k = t + 1$ and $k = 2t + 1$, both for $l = n$. pk_j refers to the public key of party p_j. The interface for the threshold signature scheme is described in Appendix F. We now describe Algorithm 7, the protocol for $BCA_{Byz,TSig}$. First, we describe the two threshold signatures that are created in the protocol:

- $\sigma_{echo,id,v}$ is a threshold signature for $k = t + 1$ proving that a non-faulty party started instance id with value v.
- $\sigma_{echo3,id,v}$ is a threshold signature for $k = 2t + 1$ proving that $t + 1$ non-faulty parties sent echo3 messages for v, and thus, by binding, no non-faulty party can output $1 - v$ from this instance.

In this setting, we are able to do away with the amplification echo and $approvedVals$ set that were used in the information theoretic setting because $\sigma_{echo,id,v}$ is a sufficient proof that some non-faulty party started instance id with value v, and it can be sent along with any message.

A non-faulty party starts the protocol by broadcasting its starting value x in an echo message along with a threshold signature share on $\langle$echo, id, $x\rangle$. Upon receiving such messages from $t + 1$ parties containing valid threshold signature shares for the same value x, a party combines them into a threshold signature, $\sigma_{echo,id,v}$, and broadcasts it in an echo2 message for v. Upon receiving such an echo2 message for a value v and valid $\sigma_{echo,idv}$ for v, a party can broadcast the same echo2 message if it hasn't already sent an echo2 message. Upon receiving echo2 messages for values v' containing valid $\sigma_{echo,id,v'}$ from $2t + 1$ parties, a party broadcasts an echo3 message:

- If the $2t + 1$ echo2 messages are for the same value v, it broadcasts $\langle$echo3, v, $\sigma_{echo,id,v}$, $ts\rangle$, where ts is a threshold signature share on (echo, id, v).
- If the $2t + 1$ echo2 messages are not for the same value v, it broadcasts $\langle$echo3, $\perp$, $\{\sigma_{echo,id,v}, \sigma_{echo,id,1-v}\}$, $\perp\rangle$.

In both cases, the σ sent in the echo3 messages are from the echo2 messages that the party received. For each echo3 message that a party receives, it checks that it contains the necessary set of proofs and that it contains a valid threshold signature share if it is for a non-$\perp$ value. Upon receiving valid echo3 messages from $2t + 1$ parties, a party outputs $\perp$ if they are not all for the same value v, or v if they are all for the same value v. In addition, if all the echo3 messages are for the same value v, a party combines the shares in the messages into a threshold signature $\sigma_{echo3,id,v}$ on (echo, id, v). In Appendix G.2 , we show how to use this proof to implement $ABA_{\frac{1}{2}}$-$EVBCA_{Byz,TSig}$, an optimized version of $ABA_{\frac{1}{2}}$-$BCA_{Byz,TSig}$ that is able to reach termination with fewer broadcasts. The pseudocode for Algorithm 7 and the proof of Theorem 6.1 are in Appendix F. Theorem 6.2 follows from the $EVBCA_{Byz,TSig}$ protocol, and the corresponding proofs can be found in Appendix G.2.

Theorem 6.1. *Algorithm 7 implements binding crusader agreement that terminates in at most 3 communication rounds and works in an asynchronous system with $n \geq 3t + 1$ parties where t parties could be Byzantine.*

Theorem 6.2. *There is an algorithm that solves asynchronous Byzantine fault tolerant agreement in expected 9 broadcasts in a system with $n \geq 3t + 1$ parties, a $2t$-unpredictable strong coin, and a threshold signature setup.*

REFERENCES

[1] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. Reaching consensus for asynchronous distributed key generation. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, PODC'21, page 363–373, New York, NY, USA, 2021. Association for Computing Machinery.

[2] Marcos Kawazoe Aguilera and Sam Toueg. Correctness proof of ben-or's randomized consensus algorithm. *Distributed Computing*, 1998.

[3] Naama Ben-David and Kartik Nayak. Brief announcement: Classifying trusted hardware via unidirectional communication. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 191–194, 2021.

[4] Michael Ben-Or. Another advantage of free choice (extended abstract) completely asynchronous agreement protocols. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 27–30, 1983.

[5] Michael Ben-Or, Danny Dolev, and Ezra N. Hoch. Brief announcement: Simple gradecast based algorithms. In Nancy A. Lynch and Alexander A. Shvartsman, editors, *Distributed Computing, 24th International Symposium, DISC 2010, Cambridge, MA, USA, September 13-15, 2010. Proceedings*, volume 6343 of *Lecture Notes in Computer Science*, pages 194–197. Springer, 2010.

[6] Gabriel Bracha. Asynchronous byzantine agreement protocols. *Information and Computation*, 1987.

[7] Christian Cachin, Klaus Kursawe, Anna Lysyanskaya, and Reto Strobl. Asynchronous verifiable secret sharing and proactive cryptosystems. In *in Proc. 9th ACM Conference on Computer and Communications Security (CCS*, pages 88–97. ACM Press, 2002.

[8] Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography. In *in Proc. 19th ACM Symposium on Principles of Distributed Computing (PODC*, pages 123–132, 2000.

[9] Christian Cachin and Luca Zanolini. From symmetric to asymmetric asynchronous byzantine consensus. *arXiv preprint arXiv:2005.08795*, 2020.

[10] Shir Cohen, Idit Keidar, and Alexander Spiegelman. Not a coincidence: Sub-quadratic asynchronous byzantine agreement whp. In *International Symposium on Distributed Computing (DISC)*, 2020.

[11] Tyler Crain. Two more algorithms for randomized signature-free asynchronous binary byzantine consensus with $t < n/3$ and $o(n^2)$ messages and $o(1)$ round expected termination. *arXiv preprint arXiv:2002.08765*, 2020.

[12] Tyler Crain, Vincent Gramoli, Mikel Larrea, and Michel Raynal. Dbft: Efficient leaderless byzantine consensus and its application to blockchains. In *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*, pages 1–8, 2018.

[13] Tyler Crain, Christopher Natoli, and Vincent Gramoli. Red belly: A secure, fair and scalable open blockchain. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 466–483, 2021.

[14] Sourav Das, Tom Yurek, Zhuolun Xiang, Andrew Miller, Lefteris Kokoris-Kogias, and Ling Ren. Practical asynchronous distributed key generation. Cryptology ePrint Archive, Report 2021/1591, 2021. https://ia.cr/2021/1591.

[15] Danny Dolev. The byzantine generals strike again. *Journal of algorithms*, 3(1):14–30, 1982.

[16] Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for byzantine agreement. *J. ACM*, 1985.

[17] Sisi Duan, Michael K. Reiter, and Haibin Zhang. Beat: Asynchronous bft made practical. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 2028–2041, New York, NY, USA, 2018. Association for Computing Machinery.

[18] Pesech Feldman and Silvio Micali. An optimal probabilistic protocol for synchronous byzantine agreement. *SIAM Journal on Computing*, 26(4):873–933, 1997.

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

[20] Matthias Fitzi, Chen-Da Liu-Zhang, and Julian Loss. A new way to achieve round-efficient byzantine agreement. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, PODC'21, page 355–362, New York, NY, USA, 2021. Association for Computing Machinery.

[21] Juan A Garay, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. Adaptively secure broadcast, revisited. In *Proceedings of the 30th annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, pages 179–186, 2011.

[22] Martin Hirt and Vassilis Zikas. Adaptively secure broadcast. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 466–485. Springer, 2010.

[23] Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for byzantine agreement. *J. Comput. Syst. Sci.*, 75(2):91–112, feb 2009.

[24] Yuan Lu, Zhenliang Lu, Qiang Tang, and Guiling Wang. Dumbo-mvba: Optimal multi-valued validated asynchronous byzantine agreement, revisited. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 129–138, 2020.

[25] Dahlia Malkhi, Michael Merritt, Michael K Reiter, and Gadi Taubenfeld. Objects shared by byzantine processes. *Distributed Computing*, 16(1):37–48, 2003.

[26] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of bft protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 31–42, New York, NY, USA, 2016. Association for Computing Machinery.

[27] Achour Mostéfaoui, Hamouma Moumen, and Michel Raynal. Signature-free asynchronous byzantine consensus with t< n/3 and o (n2) messages. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 2–9, 2014.

[28] Achour Mostéfaoui, Hamouma Moumen, and Michel Raynal. Signature-free asynchronous binary byzantine consensus with t< n/3, o (n2) messages, and o (1) expected time. *J. ACM*, 62(4):1–21, 2015.

[29] Michael O Rabin. Randomized byzantine generals. In *24th annual symposium on foundations of computer science (sfcs 1983)*, pages 403–409. IEEE, 1983.

[30] Victor Shoup. Practical threshold signatures. In *International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 207–220. Springer, 2000.
[31] Pierre Tholoniat and Vincent Gramoli. Formal verification of blockchain byzantine fault tolerance. *arXiv preprint arXiv:1909.07453*, 2019.

A PREVIOUS APPROACHES

In this section, we present an overview of Asynchronous Byzantine Agreement protocols of the past.

In [27], Mostéfaoui, Moumen, and Raynal present a protocol for Asynchronous Binary Byzantine Agreement with $O(n^2)$ messages that has optimal resilience of $t < \frac{1}{3}$ Byzantine processes. The protocol makes use of a perfect common coin as defined by Rabin in [29]. Tholoniat and Gramoli subsequently observe in [31] that the ABA protocol of [27] has a liveness violation. In order to terminate, the protocol requires a round that each party begins with the same value v. To achieve such a round, the protocol uses the common coin. At the end of each round, a party adopts the value of the random coin or a value $v \in \{0, 1\}$, such that all parties that don't adopt the coin value adopt v. Since the random coin is perfect, if all non-faulty parties adopt the value of the coin in a given round, or the coin matches the value v adopted by parties who don't adopt the coin value, then the conditions for termination are met. The flaw in the protocol is that an adversary can learn the value of the coin in a given round, and then choose v to be the opposite of the value of the coin in that round and force a non-faulty party to adopt v. If all non-faulty parties don't start the protocol with the same value, the adversary can force an infinite execution with this tactic. We refer the reader to [31] and [9] for detailed descriptions of how the adversary ensures an infinite execution.

In Algorithm 4 of [9], Cachin and Zanolini present a protocol for ABA that also uses a perfect common coin and that they claim solves the issues of [27]. Their solution makes use of a strong t-unpredictable common coin as well as the assumption of a FIFO network. However, this protocol still has a liveness issue when non-faulty parties don't all start the protocol with the same value. We construct an example with a network of 4 parties; 3 non-faulty parties (X, Y, and S), and one Byzantine party B. Let X start with 0 and Y start with 1. The starting value of S can be either 0 or 1 (we will show the attack works in either case). X and Y both broadcast their starting values in VALUE messages to all of the other parties. B sends 1 to X in a VALUE message, and 0 to Y in a VALUE message. Upon receiving the VALUE messages for 1 from B and Y, X broadcasts a VALUE message containing 1 and similarly Y broadcasts a VALUE message containing 0 upon receiving the VALUE messages for 0 from X and B. Upon receiving its own VALUE message for 1, X *abv-deliver*'s 1, and likewise Y *abv-deliver*'s 0. X and Y send AUX messages for 1 and 0, respectively. B then sends 0 to X in a VALUE message and lets X receive Y's VALUE message for 0 (along with its own VALUE message for 0), and X *abv-deliver*'s 0. Similarly, Y *abv-deliver*'s 1 after receiving the VALUE messages for 1 from itself, X and B. X and Y now send their second AUX messages for 0 and 1, respectively. B sends AUX messages for 0 and 1 to both X and Y. X and Y now satisfy the condition on line 30 in Algorithm 4 of [9], and invoke *release-coin*. X and Y receive each other's *release-coin* invocation and learn the value of the coin (as does the adversary). The coin is revealed to be s, and X and Y meet the condition on line 33 with $B = \{0, 1\}$ and propose s in the next round.

We now show how S, to whom all messages for this round have been delayed thus far, can adopt $1 - s$ for the next round without violating FIFO. Without loss of generality, assume that $s = 0$. Then the adversary lets S receive Y's VALUE message for 1. In addition, B sends a VALUE message for 1 to S. S then broadcasts a VALUE message for 1 (if it hasn't already due to 1 being its starting value). S then *abv-deliver*'s 1 and sends an AUX message for 1. The adversary also lets S receive X's VALUE messages for 0 and 1, as well as its AUX message for 1. B also sends S an AUX message for 1. At this point, S invokes *release-coin*, receives B's *release-coin* message, and subsequently reaches line 33 with $B = \{1\}$. As a result, S starts the next round with 1. Notice that had s been 1, the adversary could have forced S to reach line 33 with $B = \{0\}$, and therefore start the next round with 0, using the same tactic. Also, this tactic worked regardless of S's starting value and did not violate FIFO. S received the first 3 messages sent by X in order and only the first message sent by Y. One way to make this protocol

work would be to use a $2f$-unpredictable coin. In that case, S would only have learned the value of the coin after receiving all of the AUX messages of one of the other non-faulty parties. Since X and Y both sent AUX messages for both 0 and 1 prior to invoking *release-coin*, S would only satisfy the condition on line 33 with $B = \{0, 1\}$ and therefore would have also adopted the coin value. This solution would still require a FIFO network which, as we show in our work, is not a necessary assumption.

B CRASH TOLERANT BCA AND GBCA DEFINITIONS

Definition B.1 (Binding Crusader Agreement (BCA_{Crash})). A Binding Crusader Agreement protocol between n parties has the following guarantees:

(Agreement) If two parties decide values x and y, then either $x = y$ or at least one of the values is $\perp$.
(Validity) If all parties have the same input, then this is the only possible decision.
(Termination) All non-faulty parties eventually decide.
(Binding) Let time t be the the first time at which a party decides; at time t there is a value $b \in \{0, 1\}$ such that no party decides $1 - b$ in any extension of this execution.

Definition B.2 (Graded Binding Crusader Agreement ($GBCA_{Crash}$)). A Graded Binding Crusader Agreement protocol between n parties is one in which each party decides a value from a set of 5 ordered buckets: 0 grade 2, 0 grade 1, $\perp$ grade 0, 1 grade 1, 1 grade 2, and has the following guarantees:

(Graded Agreement) If a party decides v grade 1 or 2, no party decides v' grade 1 or 2 such that $v' = 1 - v$. Further, if a party decides v grade 2, no party decides $\perp$ grade 0.
(Validity) If all parties have the same input v, then v grade 2 is the only possible decision.
(Termination) All non-faulty parties eventually decide.
(Graded Binding) Let time t be the the first time at which a party decides; at time t there is a value $b \in \{0, 1\}$ such that no party decides $1 - b$ grade 1 or 2 in any extension of this execution.

C PROOFS FOR FRAMEWORK

C.1 Asynchronous Agreement with a Strong Coin

We now prove Theorem 3.5. To prove the theorem, we rely on a version of Lemma 3.4 that is weakened to match the weak validity property of BCA_{Crash}.

LEMMA C.1. *If all parties start the BCA_{Crash} of round r of Algorithm 1 with the same estimate value $est = v$, then all non-faulty parties commit v within a constant number of rounds.*

PROOF. The proof is very similar to that of Lemma 3.4. By the validity of BCA_{Crash}, since every party inputs v to the instance of BCA_{Crash} in round r, all parties that complete the BCA in that round decide v. If $c = v$, all participating parties commit v in this round. Otherwise, all participating parties start the next round with $est = v$. This continues until we reach a round in which $c = v$, which happens in 2 rounds in expectation. □

We are now ready to prove Theorem 3.5. Its proof is very similar to that of Theorem 3.3, except that we use Lemma C.1.

PROOF OF THEOREM 3.5. We consider each property of crash-tolerant agreement separately.

Agreement. Note that the agreement argument of Theorem 3.3 does not rely on the validity property of BCA at all, and can use Lemma C.1 instead of Lemma 3.4. Therefore, agreement holds.

Validity. If all parties have the same input, then they all start round 0 with the same estimate. Therefore, by Lemma C.1, they will commit on that input.

Termination. The termination proof is also very similar to its counterpart in Theorem 3.3. In any round in which all parties decide $\perp$ or the coin value agrees with the non-$\perp$ decision, Lemma C.1 applies. The binding property of BCA_{Crash} combined with the strong coin ensures that this happens with probability at least 50% in each round. Thus, combining this with the expected two rounds for termination in Lemma C.1, termination happens within 4 rounds in expectation. □

C.2 Asynchronous Agreement with a Weak Coin from Graded BCA

We now prove Theorem 3.6. We begin with a lemma very similar to Lemma 3.4, but even stronger.

Lemma C.2. *If all non-faulty parties start a round r of Algorithm 2 with the same estimate value $est = v$, then all non-faulty parties commit v in round r.*

Proof. By the validity of GBCA, if all non-faulty parties start their GBCA with the same value v, they all decide v with grade 2. In that case, in Algorithm 2, they all commit v regardless of the value of the coin. □

Proof of Theorem 3.6. We prove that each of the properties of asynchronous Byzantine agreement is satisfied.

Agreement. Assume that two non-faulty parties x and y commit values v and v' respectively, where $v \neq v'$. If they commit in the same round, that means that x decided v grade 2 from its GBCA and y decided v' grade 2 in the same round of GBCA, violating graded agreement.

So, assume without loss of generality that x commits first, and let the round in which x commits be round r. By graded agreement of GBCA, since x had value v grade 2 in round r, all non-faulty parties must have decided v grade 1 or 2, and therefore by Algorithm 2, all non-faulty parties start round $r + 1$ with v as their estimate. By Lemma C.2, they all commit v – a contradiction to the assumption that y commited v'.

Validity. Immediate from Lemma C.2.

Termination. Note that by the binding property of GBCA, by the time at which the first non-faulty party accesses the coin in a round r, there is some value, say v, that no non-faulty party can decide in round r (in any grade). Furthermore, by the definition of weak coin, there is a non-zero probability ϵ, that all non-faulty parties will get $1 - v$ as their coin value in round r. If this happens, then all non-faulty parties start round $r + 1$ with the same value $1 - v$, and by Lemma C.2, the protocol terminates in round $r + 1$. Therefore, the protocol terminates in a number of rounds whose expectation is upper bounded by $1 + 1/\epsilon$ for an ϵ-good coin. □

We now show that the framework presented in Algorithm 2 works for crash-tolerant agreement as well.

For the proof, we rely on a weak validity version of Lemma C.2.

Lemma C.3. *If all parties start a round of Algorithm 2 with the same estimate value $est = v$ then all parties commit v in round r.*

Proof. By the weak validity property of $GBCA_{Crash}$, if all parties start their $GBCA_{Crash}$ instance with the same value v, they will all decide v with grade 2. In that case, in Algorithm 2, they all commit v regardless of the value of the coin. □

We are now ready to prove Theorem 3.7.

Proof of Theorem 3.7. The proof is similar to the proof of Theorem 3.6.

Agreement. The agreement proof of Theorem 3.6 does not rely on the validity property, and can use Lemma C.3 instead of Lemma C.2.

Validity. Immediate from Lemma C.3.

Termination. The termination argument proceeds similarly to the termination argument of Theorem 3.6. By binding, by the time the first party accesses the coin in round r, there is some value v that cannot be decided

by any party in the $GBCA_{Crash}$ of round r. If the coin gives the same value to all parties in round r, and that value agrees with the non-$\perp$ value they decide, then Lemma C.3 applies. Therefore, the protocol terminates in a number of rounds whose expectation is upper bounded by $1 + 1/\epsilon$ for an ϵ-good coin. □

D IMPLEMENTING BINDING CRUSADER AGREEMENT PROOFS

D.1 Proofs for Asynchronous Crash Fault Tolerant Binding Crusader Agreement

Lemma D.1. *Algorithm 3 satisfies agreement.*

Proof. If a party outputs v, then they must have received at least $n - t$ echo messages containing v. Since parties send only a single echo message, by a simple quorum intersection argument, no party can output $1 - v$. □

Lemma D.2. *Algorithm 3 satisfies weak validity.*

Proof. Assume that all parties start the protocol with value v and some party p outputs $1 - v$. p must have received $\langle \text{echo}, id, 1 - v \rangle$ messages from at least $n - t$ parties, all of whom received $\langle \text{val}, id, 1 - v \rangle$ messages from at least $n - t$ parties. However, this contradicts the fact that all of the parties started with v. □

Lemma D.3. *Algorithm 3 satisfies termination within 2 communication rounds.*

Proof. By inspecting the algorithm, it is easy to see that each party only ever sends at most 2 messages. Thus, if both **upon** clauses are eventually satisfied, the lemma holds.

Every non-faulty party sends a $\langle \text{val}, id, x \rangle$ message. Therefore, every non-faulty party receives a sufficient number of val messages to send an echo message. As a result, all non-faulty parties eventually decide. □

Lemma D.4. *Algorithm 3 satisfies binding.*

Proof. In order for a party to decide non-$\perp$ v, some party must have sent $\langle \text{echo}, id, v \rangle$. By quorum intersection, if a party sends $\langle \text{echo}, id, v \rangle$, then no party can send $\langle \text{echo}, id, 1 - v \rangle$. Since a party must receive $n - t$ $\langle \text{echo}, id, v \rangle$ to decide non-$\perp$ v, binding is achieved once the first party decides. □

Lemmas D.1, D.2, D.3, and D.4 together imply Theorem 4.1. Plugging this into Theorem 3.3, we get the proof for Theorem 4.2.

Proof of Theorem 4.2. Consider the $AA_{\frac{1}{2}}$-BCA_{Crash} protocol. The adversary has two strategies in a given round:

(1) All parties output $\perp$ from BCA_{Crash}
(2) Some parties output a non-$\perp$ value from BCA_{Crash}

If the adversary chooses the first strategy, then the parties adopt the same value and decide in the next round of $AA_{\frac{1}{2}}$ where the coin is equal to that value, so 3 rounds in expectation.

If the adversary chooses the second strategy, it takes in expectation two rounds of $AA_{\frac{1}{2}}$ for the coin to equal the non-$\perp$ decision value. Once this happens, some party must have commited.

The first strategy takes 3 rounds in expectation for a party to commit, while the second strategy takes 2 rounds in expectation. Once a party commits, they broadcast a "commit" message to all of the other parties and terminate. All of the other parties also broadcast this message and terminate once they receive it. Since there are 2 broadcasts per round of BCA_{Crash}, the protocol terminates in an expected 7 broadcasts. □

E IMPLEMENTING GRADED BCA PROOFS

E.1 Asynchronous Crash Fault Tolerant Graded Binding Crusader Agreement

Lemma E.1. *Algorithm 5 satisfies graded agreement.*

Proof. First, we prove that if a party outputs v grade 1 or 2, no party outputs v' grade 1 or 2 such that $v' = 1 - v$. If a party decides v (grade 1 or 2), they received at least one $\langle \text{echo2}, id, v \rangle$ message. In order to send $\langle \text{echo2}, id, v \rangle$, a party must have received $\langle \text{echo}, id, v \rangle$ messages from at least $n - t$ parties. Since a party sends an echo message for a single value v, the statement follows from a quorum intersection argument.

Next, we prove that if a party outputs v grade 2, no non-faulty party outputs $\perp$ grade 0. If a party outputs v grade 2, they received $\langle \text{echo2}, id, v \rangle$ messages from at least $n - t$ parties. Therefore, every party will see at least one $\langle \text{echo2}, id, v \rangle$ message. □

Lemma E.2. *Algorithm 5 satisfies termination within 3 communication rounds.*

Proof. It is clear from examining the protocol that parties broadcast at most 3 messages: a val message, an echo message, and an echo2 message.

Since there are at least $n - t$ non-faulty parties, every non-faulty party receives $\langle \text{val}, id, * \rangle$ messages from at least $n - t$ parties and sends an echo message. Every party therefore receives echo messages from at least $n - t$ parties, and all non-faulty parties send echo2 messages. Finally, every party receives at least $n - t$ echo2 messages, and all non-faulty parties decide a value. □

Lemma E.3. *Algorithm 5 satisfies validity.*

Proof. Assume that all parties start with v. We will prove that no party outputs $1 - v$ grade 1 or 2, v grade 1, or $\perp$ grade 0. In order to output such a value, a party must receive an echo2 message containing a value other than v. In order for a party to send such an echo2 message, a party must have sent an echo message for a value other than v. This means a party must have started with a value other than v, contradicting our original statement. Therefore, by termination, all the parties decide v grade 2. □

Lemma E.4. *Algorithm 5 satisfies binding.*

Proof. At the time at which the first party decides, at least $n - t$ parties must have sent echo2 messages. In order to output a value v grade 1 or 2, a party must receive at least one $\langle \text{echo2}, id, v \rangle$ message. Since a party sends only a single echo2 message with a non-$\perp$ value, by the time the first $n - t$ parties send echo2 messages, there must be a value $v' \in \{0, 1\}$ such that no party can output v' grade 1 or 2. □

Proof of Theorem 5.1. The Theorem is implied by Lemmas E.1, E.2, E.3 and E.4. □

Proof of Theorem 5.2. Consider the AA_ϵ-$GBCA_{Crash}$ protocol. If the value of the coin for all parties is equal to the non-$\perp$ decision value of the parties in that round, or they all output $\perp$, then Lemma E.3 applies in the next round and all of the non-faulty parties commit. In expectation, it takes $1/\epsilon$ rounds for the parties to adopt the same value. Once they do, there is an additional round for them to decide that value, grade 2. There are 3 broadcasts per round and parties send a final broadcast once they commit. This gives us a total of $3/\epsilon + 3 + 1 = 3/\epsilon + 4$ broadcasts. □

E.2 Asynchronous Byzantine Fault Tolerant Graded Binding Crusader Agreement

We now prove that Algorithm 6 implements Graded Binding Crusader Agreement. We first start with an important lemma.

Lemma E.5. *If no non-faulty party terminates the protocol, then the $approvedVals$ sets of all non-faulty parties are eventually equal.*

Proof. A non-faulty party adds value v to its $approvedVals$ set if it receives $\langle \text{echo}, id, v\rangle$ messages from at least $n - t$ distinct parties. At least $t + 1$ of these messages had to have been from non-faulty parties, and therefore will be received by all non-faulty parties. As a result, all non-faulty parties send amplification echos for v (Lines 3-4), and all parties receive a sufficient number of echo messages for v to add v to their $approvedVals$ sets. □

Lemma E.6. *Algorithm 6 satisfies graded agreement.*

Proof. First we prove that if a non-faulty party outputs non-$\perp$ value v grade 1 or 2, then no non-faulty party outputs $1 - v$ grade 1 or 2. Assume that a non-faulty party outputs v grade 1. Then they must have seen $t + 1$ $\langle \text{echo4}, id, v\rangle$ messages, at least one of which was from a non-faulty party. The statement therefore follows from Lemma 4.4 and the fact that outputting v grade 2 requires more than one non-faulty party to send $\langle \text{echo4}, id, v\rangle$ (and likewise for $1 - v$).

Next, we prove that if a non-faulty party outputs non-$\perp$ value v grade 2, no non-faulty party outputs $\perp$ grade 0. In order to output v grade 2, a party must receive $\langle \text{echo5}, id, v\rangle$ messages from at least $n - t$ parties. By quorum intersection, and the fact that each non-faulty party broadcast a single echo5 message, no non-faulty party can receive the required $n - t$ $\langle \text{echo5}, id, \perp\rangle$ messages to decide $\perp$ grade 0. □

Lemma E.7. *Algorithm 6 satisfies validity.*

Proof. Assume all non-faulty parties start the protocol with v. By Lemma 4.5, all non-faulty parties send $\langle \text{echo4}, id, v\rangle$. Next, we show that no non-faulty party adds $1 - v$ to its $approvedVals$ set. To show this, we prove that no non-faulty party ever sends an $\langle \text{echo}, id, 1 - v\rangle$ message. In order to send an $\langle \text{echo}, id, 1 - v\rangle$ message, a non-faulty party must receive $\langle \text{echo}, id, 1 - v\rangle$ from $t + 1$ parties. This cannot happen if all non-faulty parties start with v. Therefore, non-faulty parties will only send an echo5 message after receiving $n - t$ $\langle \text{echo4}, id, v\rangle$ messages, and will therefore all send $\langle \text{echo5}, id, v\rangle$. Since no non-faulty party adds $1 - v$ to its $approvedVals$, all non-faulty parties decide only upon receiving $n - t$ $\langle \text{echo5}, id, v\rangle$ messages and output v. □

Lemma E.8. *Algorithm 6 satisfies termination within 6 communication rounds.*

Proof. It is clear from examining the protocol that a party sends at most two echo messages and at most one echo2, one echo3, one echo4, and one echo5 message.

By Lemma 4.7, all non-faulty parties send echo4 messages. If all non-faulty parties send $\langle \text{echo4}, id, v\rangle$ for the same non-$\perp$ value v, then all non-faulty parties receive $\langle \text{echo4}, id, v\rangle$ messages from a sufficient number of parties to send $\langle \text{echo5}, id, v\rangle$. All non-faulty parties then receive $\langle \text{echo5}, id, v\rangle$ from a sufficient number of parties to decide v.

Next, we consider the case where all non-faulty parties do not send $\langle \text{echo4}, id, v\rangle$ for the same non-$\perp$ value v. Assume that this is the case. Then, by Lemma 4.4, the only other value non-faulty parties could have sent is $\langle \text{echo4}, id, \perp\rangle$. This non-faulty party must have its $|approvedVals| > 1$. By Lemma E.5, $|approvedVals| > 1$ for all non-faulty parties eventually, and all non-faulty parties accept this echo4 message. As a result, all non-faulty parties send echo5 messages. If a non-faulty party sends $\langle \text{echo5}, id, v\rangle$, they received $\langle \text{echo4}, id, v\rangle$ from at least $n - t$ parties, and every parties is guaranteed to receive $\langle \text{echo4}, id, v\rangle$ from at least $t + 1$ parties. If a non-faulty party sends $\langle \text{echo5}, id, \perp\rangle$, again by Lemma E.5, all non-faulty parties accept this echo5 message. As a result, all parties receive a sufficient number of valid echo5 messages and decide a value. □

Lemma E.9. *Algorithm 6 satisfies binding.*

Proof. In order for a non-faulty party to decide v or $v\perp$, at least one non-faulty party must send $\langle$echo4, id, $v\rangle$. The lemma therefore follows from Lemma 4.9. □

Proof of Theorem 5.3. The Theorem is implied from Lemmas E.6, E.7, E.8 and E.9. □

Proof of Theorem 5.4. Consider the AA_ϵ-$GBCA_{Byz}$ protocol. If the value of the coin for all non-faulty parties in a given round is equal to the non-$\perp$ decision value of the non-faulty parties in that round, or they all output $\perp$, then Lemma E.7 applies in the next round and all the non-faulty parties commit. In expectation it takes $1/\epsilon$ rounds until this happens, and there are at most 6 broadcasts per round until it does. Since all non-faulty parties start the round in which they commit with the same value, there are 5 broadcasts in that round. There is one final broadcast after all the non-faulty parties decide to ensure termination. □

F BINDING CRUSADER AGREEMENT WITH THRESHOLD SIGNATURES

We use the following interface for the threshold signature scheme:

- threshold-sign$_i(m)$: produces signature share by p_i on message m.
- share-validate(m, s_j, pk_j): validates that signature share s_j was produced by p_j on m.
- threshold-combine(m, S): combines a set S of signature shares from distinct parties for message m into an $O(1)$-sized signature where $|S| \geq k$ and $\forall s_j \in S$, share-validate$(m, s_j, pk_j) = true$.
- threshold-verify(m, σ): returns true if σ was a result of computing threshold-combine(m, S) where $|S| \geq k$ and share-validate$(m, s_j, pk_j) = true, \forall s_j \in S$.

Algorithm 7: Asynchronous Binding Crusader Agreement with Threshold Signatures instance id ($BCA_{Byz,TSig}$)

Input: x

$est = x$
$sentEcho2 = false$, $sentEcho3 = false$, $pendingEcho = \{\}$, $pendingEcho2 = \{\}$, $pendingEcho3 = \{\}$, $vals = \{\}$
send $\langle$echo, est, threshold-sign$_i$(echo, id, est)$\rangle$ to all
upon receiving a message $\langle$echo, v, $ts\rangle$ from p_j for the first time such that share-validate((echo, id, v), ts, p_j) = true
  add (v, ts) to $pendingEcho$
  if there exists a single value v' such that the set M of $\{(v', ts') | (v', ts') \in pendingEcho\}$ is of size at least $t + 1$ and $sentEcho2 = false$ **then**
    let S be the set $\{s | (*, s) \in pendingEcho\}$
    $\sigma_{\text{echo},id,v'} \leftarrow$threshold-combine((echo, id, v'), S)
    send $\langle$echo2, v', $\sigma_{\text{echo},id,v'}\rangle$ to all, $sentEcho2 = true$
upon receiving a message $\langle$echo2, v, $\sigma\rangle$ from p_j for the first time such that threshold-verify((echo, id, v), σ) = $true$
  if $sentEcho2 = false$ **then**
    send $\langle$echo2, v, $\sigma\rangle$ to all, $sentEcho2 = true$
  add (v, σ) to $pendingEcho2$
  if $|pendingEcho2| \geq n - t$ and $sentEcho3 = false$
    let M be the set $\{v | (v, *) \in pendingEcho2\}$ and let S be the set $\{s | (*, s) \in pendingEcho2\}$
    if $|M| > 1$ **then**
      send $\langle$echo3,$\perp$, S, $\perp\rangle$ to all
    else send $\langle$echo3, v, S, threshold-sign$_i$(echo3, id, v)$\rangle$ to all, where v is the single value in M
    $sentEcho3 = true$
upon receiving a message m where m=$\langle$echo3, v, S, $ts\rangle$ from p_j for the first time such that either $v = \perp$ or share-validate((echo3, id, v), ts, p_j)=$true$
  if $v = \perp$ **then** $vals = \{0, 1\}$
  else $vals = \{v\}$
  if $\forall v' \in vals$ there exists $\sigma' \in S$ s.t. threshold-verify((echo, id, v'), σ') = $true$ **then**
    add (v, ts) to $pendingEcho3$
    if $|pendingEcho3| \geq n - t$ **then**
      let M be the set of values $\{v | (v, *) \in pendingEcho3\}$
      if $|M| > 1$ **then** decide $\perp$
      else
        let S be the set $\{s | (*, s) \in pendingEcho3\}$, and let v be the single value in M
        $\sigma_{\text{echo3},id,v} \leftarrow$threshold-combine((echo3, id, v), S)
        decide v

We now prove Theorem 6.1.

Proof of Theorem 6.1. The theorem follows from lemmas F.1-F.5. □

Lemma F.1. *Algorithm 7 satisfies agreement.*

Proof. Assume that p_i decides non-$\perp$ value v_i. Then p_i received $\langle$echo3, v_i, S, $ts\rangle$ from at least $n-t$ parties p_k such that share-validate((echo3, id, v_i), ts, p_k) = $true$. Since a non-faulty party sends a single echo3 message, by quorum intersection either $v_j = \perp$ or $v_i = v_j$. □

Lemma F.2. *Algorithm 7 satisfies validity.*

Proof. Assume, for a contradiction, that all non-faulty parties input v to the protocol for instance id, and some non-faulty party p_i decides $v' \neq v$ (v' must be either $1-v$ or $\perp$). p_i must have received a message $\langle$echo3, v', S', $ts\rangle$ from at least one party such that there is a $\sigma' \in S'$ and threshold-verify((echo, id, v'), σ') = $true$. For such a σ' to exist, that would require at least $t+1$ parties p_j to have sent $\langle$echo, v', threshold-sign$_j$(echo, id, v')$\rangle$ messages, at least one of which would have to be non-faulty. Non-faulty parties only send a single echo message, and only for the value that they started with. Therefore, we have arrived at a contradiction. □

Lemma F.3. *Algorithm 7 satisfies termination within 3 communication rounds.*

Proof. It is clear from examining the protocol that a party sends at most three messages: echo, echo2, and echo3.

Since we are considering the binary case, there must be some value v such that at least $t+1$ non-faulty parties start with v. Therefore, eventually every non-faulty party receives at least $t+1$ $\langle$echo, v, $ts\rangle$ from non-faulty parties p_i, where ts is p_i's output from running threshold-sign$_i$(echo, id, v). If they haven't already, every non-faulty party then sends an echo2 message containing a valid threshold signature σ on (echo, id, v). Every party then receives echo2 messages containing valid threshold signatures from at least $n-t$ parties and therefore has at least $n-t$ entries in its *pendingEcho2* set. Each non-faulty party then sends an echo3 message accordingly, containing the set of threshold signatures needed for the value included in its message, along with a threshold signature share if it sends a non-$\perp$ value. Finally, each party receives at least $n-t$ such echo3 messages and decides a value. □

Lemma F.4. *If a non-faulty party sends* $\langle$echo3, v, *, *$\rangle$, *where* $v \neq \perp$ *and another non-faulty party sends* $\langle$echo3, v', *, *$\rangle$, *either* $v' = v$ *or* $v' = \perp$.

Proof. Assume, for a contradiction, that non-faulty party p_i sends $\langle$echo3, v_i, *, *$\rangle$ and non-faulty party p_j sends $\langle$echo3, v_j, *, *$\rangle$, $v_i \neq v_j$, and both values are non-$\perp$. Then p_i must have at least $n-t$ $(v_i, *)$ in its *pendingEcho2* set and p_j must have at least $n-t$ $(v_j, *)$ in its *pendingEcho2* set. However, due to quorum intersection and the fact that a non-faulty party sends a single echo2 message, this is not possible. Thus, we have arrived at contradiction. □

Lemma F.5. *Algorithm 7 satisfies binding.*

Proof. Let p_i be the first non-faulty party who decides. Consider the set of non-faulty parties of size at least $n-2t \geq t+1$ whose values sent in their echo3 messages are in p_i's *pendingEcho3* set at the time at which it decides. If all of these non-faulty parties sent $\langle$echo3, $\perp$, *, *$\rangle$, then every non-faulty party has at least one such message in their *pendingEcho3* set and therefore outputs $\perp$. If instead at least one of these non-faulty parties sent $\langle$echo3, b, *, *$\rangle$ where $b \neq \perp$, then, by Lemma F.4, no non-faulty party sends $\langle$echo3, $1-b$, *, *$\rangle$ and no non-faulty party can decide $1-b$. □

Lemma F.6. *The* $AA_{\frac{1}{2}}$*-*$BCA_{Byz,TSig}$ *protocol takes, in expectation, 13 rounds to terminate with a strong* $2t$*-resilient common coin.*

Proof. In an instance of $BCA_{Byz,TSig}$, parties perform exactly three broadcasts. Since the common coin has degree $2f$, the messages to reveal the coin can be sent with echo3 messages. It takes, in expectation, 2 rounds of $AA_{\frac{1}{2}}$-$BCA_{Byz,TSig}$ for all non-faulty parties to adopt the same value v and an additional 2 rounds in expectation

until the coin value is again v. As a result, the protocol takes 12 broadcasts until all non-faulty parties commit, in expectation. Since every non-faulty party obtains a $\sigma_{echo3,id,v}$ when they decide v in instance id, they can forward this proof to all of the other parties and terminate because parties can decide v after receiving it. As a result, the protocol requires a single additional message for all parties to terminate, resulting in 12+1=13 broadcasts in expectation. □

G EXTERNALLY VALID BINDING CRUSADER AGREEMENT

In previous sections, we presented protocols for Asynchronous Agreement and Binding Crusader Agreement in a decoupled fashion. Now that we have explained the simple framework for AA based on BCA, we show a series of modifications to make AA protocols terminate with fewer broadcasts. When using BCA within an AA protocol, we can take into account the context of the AA protocol within the BCA execution (such as the value of the coin of the previous round) to reduce the number of broadcasts in each round. In this section, we present these modifications and prove their correctness. These optimizations rely on the fact that in certain cases, the proof that a value or message is "valid" exists, and is guaranteed to be received by all parties. In these conditions, parties can omit sending certain messages for this value in the next round, or send certain messages early, since all parties are guaranteed to receive proof that this value or message is "valid" and act accordingly in that round.

To illustrate an example, consider the $AA_{\frac{1}{2}}$-BCA_{Byz} protocol. If a party has proof that some non-faulty party started round r of the protocol with 0 (i.e. by receiving $t + 1$ echo messages for 0) and some non-faulty party started round r with 1, and the value of the coin in round r is 0, then the party knows that it is possible that some non-faulty party could have decided $\perp$ within the BCA_{Byz} instance of round r and updated its est to 0, the value of the coin. In addition, it knows that 0 is a "safe value" for the subsequent round because no non-faulty party would have commited 1 in round as it is not the coin value. Non-faulty parties that adopt 0 in round r as a result of deciding $\perp$ from the instance of BCA_{Byz} can therefore omit sending echo messages for 0 in round $r + 1$. Note that the existence of such a proof that 0 is a valid value *doesn't guarantee that a non-faulty party did output that value from* BCA_{Byz}; it only means that they *could have*. The modified protocols therefore no longer satisfy the original validity properties of BCA and GBCA as defined in Definition 3.1 and Definition 3.2. For this reason, we define a new validity property, *external validity*, that is a property of *Externally Valid Binding Crusader Agreement* (EVGBCA). External validity captures that a value is "externally valid" if a proof of its validity exists, even if no non-faulty party started with that value.

We first present a definition for Externally Valid Binding Crusader Agreement (EVBCA) and prove that when a protocol that implements EVBCA is plugged into $AA_{\frac{1}{2}}$ respectively, the result is a protocol implementing AA. In section G.1, we should how to get Externally Valid Binding Crusader Agreement for Byzantine faults ($EVBCA_{Byz}$) from BCA_{Byz}. We then prove that $EVBCA_{Byz}$ implements EVBCA. In section G.2, we show the same for Externally Valid Binding Crusader Agreement with threshold signatures for Byzantine faults ($EVBCA_{Byz,TSig}$) from $BCA_{Byz,TSig}$.

We first define Externally Valid Binding Crusader Agreement (EVBCA) with an external function called Ext-Valid() that returns whether a value is valid or not.

Definition G.1. (Externally Valid Binding Crusader Agreement (EVBCA)) An Externally Valid Binding Crusader Agreement protocol between n parties has the following guarantees:

> **(Agreement)** If two non-faulty parties output values x and y, then either $x = y$ or at least one of the values is $\perp$.
> **(External Validity)** If Ext-Valid(v)=$true$ and Ext-Valid($1 - v$)=$false$, all non-faulty parties decide v.
> **(Termination)** All non-faulty parties eventually decide.

(Binding) Let t be the first time at which a party that is non-faulty at the time of deciding decides. At time t there is a value $b \in \{0, 1\}$ such that no non-faulty party decides $1 - b$ in any extension of this execution.

Next, we define an externally valid value for the $AA_{\frac{1}{2}}$ in the context of the $AA_{\frac{1}{2}}$ protocol.

Definition G.2. (Externally Valid Value for $AA_{\frac{1}{2}}$) A value v is externally valid in round r of the $AA_{\frac{1}{2}}$ protocol if either of the following conditions are met:

(1) A non-faulty party started round r of the protocol with v.
(2) v was externally valid in round $r - 1$ and no non-faulty party committed $1 - v$ in round $r - 1$.

We now show that a protocol implementing EVBCA can be plugged into the $AA_{\frac{1}{2}}$ protocol (Algorithm 1) to solve AA.

Theorem G.3. *If we plug an implementation of* EVBCA *into* BCA *and* CommonCoin *is a correct implementation of a strong coin, then Algorithm 1 solves asynchronous Byzantine agreement in any system in which* $n \geq 3t + 1$ *and terminates in 4 rounds in expectation against an adaptive adversary.*

Before proving Theorem G.3, we start with a useful lemma.

Lemma G.4. *If all non-faulty parties start round* r *with the same value* $est = v$*, and* Ext-Valid$(1 - v)$=$false$*, then all non-faulty parties commit* v *within a constant number of rounds.*

Proof. By the external validity property of EVBCA, every non-faulty party outputs v from EVBCA and therefore sets $est = v$ in round r. Since no non-faulty party starts round $r + 1$ with $1 - v$ and $1 - v$ was not externally valid in round r, Ext-Valid$(1 - v)$=$false$ in round $r + 1$. This continues for every round by the definition of external validity. In round $r' \geq r$ where v is equal to the value returned by CommonCoin for round r', all non-faulty parties commit v. In expectation, this happens in 2 rounds since CommonCoin has a 50% probability of returning v in each round. □

Proof of Theorem G.3. We prove that each of the properties of asynchronous Byzantine agreement is satisfied.

Agreement. A non-faulty party p only commits a value in round r if the coin agreed with this value and it was the output of p's EVBCA in round r. Let p be the first non-faulty party that commits, let r be the round in which it commits, and let v be its commit value. Note that in round r no other value can be commited since the commit value is always the same as the coin of that round and the coin is strong. By the agreement property of EVBCA, every non-faulty party decides v or $\perp$ from EVBCA, and therefore all non-faulty parties set $est = v$ at the end of this round.

By the definition of an externally valid value, Ext-Valid$(1 - v)$=$false$ in round $r + 1$. By the external validity property of EVBCA, in subsequent rounds of $AA_{\frac{1}{2}}$ no non-faulty party ever decides $1 - v$ or $\perp$ from EVBCA and $1 - v$ never becomes externally valid. Therefore, no non-faulty party ever decides $1 - v$ in round $r' > r$.

Validity. If all non-faulty parties start the protocol with v, then $1 - v$ cannot be externally valid by the definition of an externally valid value. Therefore, by Lemma G.4 all non-faulty parties will commit v.

Termination. Note that if in any round the coin value is equal to the non-$\perp$ decision value of non-faulty parties or all of the non-faulty parties output $\perp$, Lemma G.4 applies in the next round. Furthermore, by the binding property of EVBCA, the adversary is bound to the non-$\perp$ value b that can possibly be output by a non-faulty party in a round r (if there is one), by the time the first non-faulty party finishes its EVBCA in round r. In particular, this must happen before the coin value is revealed in any coin of degree t or larger. Therefore, in each round, there is at least a 50% chance that the value b to which the adversary is bound will be the same as the coin or $\perp$. □

G.1 Implementing Externally Valid Binding Crusader Agreement

We now present modifications that can be applied to the BCA_{Byz} protocol of Algorithm 4 to obtain $EVBCA_{Byz}$, a protocol that implements EVBCA. When used with a $2t$ degree strong common coin in the $AA_{\frac{1}{2}}$ protocol of Algorithm 1, the $EVBCA_{Byz}$ requires at most 3 broadcasts per round, and a minimum of two broadcasts per round, for every round after the first round. We first present the definition of an externally valid value that is specific to $AA_{\frac{1}{2}}$-BCA_{Byz}. We then prove that this definition satisfies the definition of an externally valid value for the $AA_{\frac{1}{2}}$ protocol in Definition G.2. After that, we present the modifications to get $EVBCA_{Byz}$ from BCA_{Byz}. Finally, we prove that $EVBCA_{Byz}$ implements EVBCA.

Definition G.5. (Externally Valid Value for $AA_{\frac{1}{2}}$-BCA_{Byz}) A value v is **externally valid** for the in round r of the $AA_{\frac{1}{2}}$-BCA_{Byz} protocol if either of the following conditions are met:

(1) v is eventually in the $approvedVals$ set of all non-faulty parties in round $r-1$ and v is the value of the common coin in round $r-1$.
(2) Some non-faulty party starts round r of the protocol with value v.

Lemma G.6. *Externally Valid Value for $AA_{\frac{1}{2}}$-BCA_{Byz} (Definition G.5) satisfies the definition of an Externally Valid Value for $AA_{\frac{1}{2}}$ in Definition G.2.*

Proof. We use a proof by induction. Consider round 0 of the $AA_{\frac{1}{2}}$-BCA_{Byz} protocol. If all non-faulty parties start round 0 with value v, no non-faulty party adds $1-v$ to their $approvedVals$ set in round 0, and no non-faulty party decides $1-v$ or $\perp$ from the BCA_{Byz} of that round. By validity of BCA_{Byz}, no non-faulty party ever starts a round with $1-v$ or adds $1-v$ to their $approvedVals$ set. □

We now present the set of modifications that are applied to BCA_{Byz} to get the $EVBCA_{Byz}$ protocol.

(1) In round r, p_i automatically adds v to its $approvedVals$ set if p_i's $approvedVals$ set of round $r-1$ contains v and v was the value of the common coin in round $r-1$.
(2) Upon adding value v to its $approvedVals$ set in round r, if p_i did not already send an echo2 message in round r, p_i sends $\langle \text{echo2}, r, v\rangle$ to all parties.
(3) If p_i outputs $\perp$ from the BCA_{Byz} of round r and the coin value in round r is c, p_i does not send an echo message in round $r+1$ and directly broadcasts an $\langle \text{echo2}, r+1, v\rangle$ message.
(4) Upon outputting v from the BCA_{Byz} of round r, if the common coin of round r also equals v, then p_i automatically broadcasts the messages $\langle \text{echo2}, r+1, v\rangle$ and $\langle \text{echo3}, r+1, v\rangle$ together in round $r+1$. Note that this does not hinder liveness because all non-faulty parties automatically add v to their $approvedVals$ set of round r by optimization 1 and Lemma G.8.

We now prove that the $EVBCA_{Byz}$ protocol implements EVBCA.

Theorem G.7. *The $EVBCA_{Byz}$ protocol implements* EVBCA.

Proof. The theorem follows from Lemmas G.9, G.10, G.11, and G.14. □

Before proving that each property of EVBCA is satisfied we start with a useful lemma.

Lemma G.8. *If non-faulty party p_i adds v to its $approvedVals$ set of round r, then eventually v is in the round r $approvedVals$ set of all non-faulty parties.*

Proof. We use a proof by induction. Consider the $EVBCA_{Byz}$ in round $r=0$ of the $AA_{\frac{1}{2}}$-$EVBCA_{Byz}$ protocol. The only way for a non-faulty party to add a value v to its $approvedVals$ set in this round is if it receives at least $n-t$ $\langle \text{echo}, r, v\rangle$ messages. At least $t+1$ of these messages must have been sent by non-faulty parties and are

therefore received by all parties. As a result, all non-faulty parties send $\langle \text{echo}, r, v \rangle$ messages, and all non-faulty parties add v to their $approvedVals$ set.

Now, assume the lemma is true for round $r'' > 0$, consider round $r' = r'' + 1$, and let c be the value of the common coin in round r''. If a non-faulty party adds a value to its $approvedVals$ set as a result of receiving echo messages for it from $n - t$ parties in round r', then by the same argument used for round 0, the value is eventually added to the round r' $approvedVals$ sets of all non-faulty parties. Otherwise, if a party adds c to its round r' $approvedVals$, this was because it had c in its round r'' $approvedVals$ set. Since the common coin is strong and by the assumption for round r'', all non-faulty parties add c to their round r' $approvedVals$ sets. □

Lemma G.9. *The $EVBCA_{Byz}$ protocol satisfies agreement.*

Proof. This follows from Lemma 4.4. □

Lemma G.10. *The $EVBCA_{Byz}$ protocol satisfies external validity.*

Proof. Consider a round r in which v is the only externally valid value. This means that all non-faulty parties start the round with value v and $1 - v$ was not the value of the coin in the previous round, and/or it is never in their round $r - 1$ $approvedVals$ sets. Then the only way that a non-faulty party can add $1 - v$ to their round r $approvedVals$ is if they receive echo messages for $1 - v$ in round r from at least $n - t$ parties. This cannot happen since no non-faulty party starts round r with $1 - v$. As a result, v is the only value that can be decided by non-faulty parties in round r. The rest of the proof follows from Lemma G.11. □

Lemma G.11. *The $EVBCA_{Byz}$ protocol satisfies termination.*

Proof. There are two possible cases to consider for a given round r of the protocol:

(1) There is a single externally valid value v in round r.
(2) There are two externally valid values in round r.

In case 1, if v was not the value of the coin in the previous round (meaning all non-faulty parties start round r with v and therefore send $\langle \text{echo}, r, v \rangle$ messages), termination follows from Lemma 4.7.

In case 1, if v was the value of the coin in the previous round, all non-faulty parties add v to their $approvedVals$ sets and directly send an echo2 message for v. All non-faulty parties then receive a sufficient number of echo2 messages for v to send echo3 messages for v. Finally, all non-faulty parties decide v.

In case 2, first we prove that all non-faulty parties send echo2 messages. If the value of the coin, c, in round $r - 1$ is added to the $approvedVals$ set of all non-faulty parties in round $r - 1$, then eventually all non-faulty parties add c to their round r $approvedVals$ set and send echo2 messages for c in round r. If this is not the case, there must be some value such that at least $t + 1$ non-faulty parties start round r with that value, and all non-faulty parties eventually receive a sufficient number of echo messages for it to send an echo2 message. Then termination follows from 4.7. In either case, since a non-faulty party adds a value to their $approvedVals$ prior to sending an echo2 message for it, by Lemma G.8, termination follows.

□

Before proving that $EVBCA_{Byz}$ satisfies binding, we prove that there is at most one non-$\perp$ value v for which non-faulty parties send echo3 messages in a given round.

Lemma G.12. *If a non-faulty party sends an echo3 message for value v in round r, then no non-faulty party sends an echo3 message for $1 - v$ in round r.*

Lemma G.13. *We use a proof by contradiction. Let p_i and p_j be non-faulty parties that send echo3 messages for v_i and v_j respectively in round r. If neither party sent an early echo3 message as a result of deciding the value of the coin of round $r - 1$ in the* EVBCA *of round $r - 1$ (optimization 4), then the lemma follows from Lemma 4.8.*

Assume p_i sent an echo3 message for v_i as a result of optimization 4. We will show that it must be the case that $v_j = v_i$. By agreement (Lemma G.9), all non-faulty parties decide v_j or $\perp$ from the $EVBCA_{Byz}$ of round $r-1$. Therefore, since the common coin is strong, all non-faulty parties start round r with $est = v_i$. By the definition of an externally valid value, $1 - v_i$ is not externally valid. If $v_j = 1 - v_i$, p_j must have added $1 - v_i$ to its approvedVals set of round r. Since $1 - v_i$ is not value of the common coin of round $r - 1$ and no non-faulty party starts round r with $est = 1 - v_i$, $v_j = v_i$.

Lemma G.14. *The $EVBCA_{Byz}$ protocol satisfies binding.*

Proof. This follows from Lemma G.12 and Lemma 4.9. □

Lemma G.15. *The $ABA_{\frac{1}{2}}$-$EVBCA_{Byz}$ protocol solves asynchronous Byzantine agreement in expected 13 broadcasts in a system with $n \geq 3t + 1$ parties and a degree $2t$ strong coin against an adaptive adversary if* `CommonCoin` *is a correct implementation of a strong common coin.*

Proof. In expectation it takes 2 rounds of the $AA_{\frac{1}{2}}$-$EVBCA_{Byz}$ protocol until the coin value is equal to the non-$\perp$ decision value of non-faulty parties. Once all non-faulty parties have adopted the same value, it takes an additional two rounds in expectation for the coin to be equal that value again so they may all decide. As a result, the $AA_{\frac{1}{2}}$-$EVBCA_{Byz}$ protocol takes in expectation 4 rounds for all non-faulty parties to decide. Next, we must count the number of broadcasts that occur in each of the 4 rounds.

In the $AA_{\frac{1}{2}}$-$EVBCA_{Byz}$ protocol, in every round after the first round, at most one value, the value opposite the coin value of the previous round, is echoed by non-faulty parties. If all non-faulty parties start a round with the same value and this value is the same as the value of the coin of the previous round, non-faulty parties don't send any echo messages. There are two rounds in which non-faulty parties don't start with the same value and two rounds that they do start with the same value in expectation. As a result, in expectation there are 4 broadcasts in the first round, 2 broadcasts in the round that occurs after all non-faulty parties adopt the coin value, 3 broadcasts in the remaining two rounds, and a final broadcast to ensure that all non-faulty parties terminate totalling 13 broadcasts.

Note that since the coin is of degree $2t$, and binding is achieved once the first $t+1$ non-faulty parties send echo3 messages, the reveal coin messages can be sent with the echo3 messages so that they don't require additional broadcasts. □

G.2 Implementing Externally Valid Binding Crusader Agreement with Threshold Signatures

In section 6, we presented $BCA_{Byz,TSig}$, a protocol for binding crusader agreement that, when plugged into $AA_{\frac{1}{2}}$ to get $AA_{\frac{1}{2}}$-$BCA_{Byz,TSig}$, implements Asynchronous Byzantine Agreement with a strong coin and threshold signatures. As shown in Lemma F.6, the $AA_{\frac{1}{2}}$-$BCA_{Byz,TSig}$ protocol takes an expected 13 rounds of broadcast to terminate with a $2t$ degree strong coin. In this section, we describe two optimizations that can be applied to $BCA_{Byz,TSig}$ to obtain $EVBCA_{Byz,TSig}$ so that $AA_{\frac{1}{2}}$-$EVBCA_{Byz,TSig}$ requires 9 rounds of broadcast to terminate in expectation when used with a degree $2t$ strong coin.

We first define an externally valid value in the context of the $AA_{\frac{1}{2}}$-$BCA_{Byz,TSig}$ protocol and show that it satisfies Definiton G.2.

Definition G.16. (Externally Valid Value for $AA_{\frac{1}{2}}$-$BCA_{Byz,TSig}$) A value v is **externally valid** in round r of the $AA_{\frac{1}{2}}$-$BCA_{Byz,TSig}$ protocol if any of the following conditions are satisfied:

(1) A non-faulty party starts round r with value v.
(2) There is a $\sigma_{\text{echo3},r',v}$ from round $r' \leq r$ where v was the value of the common coin in round r'.
(3) There is a $\sigma_{\text{echo3},r-1,v}$ from round $r-1$ where v was not the value of the common coin in round $r-1$.

Lemma G.17. *Externally Valid Value for $AA_{\frac{1}{2}}$-$BCA_{Byz,TSig}$ satisfies the definition of Externally Valid Value for $AA_{\frac{1}{2}}$ in Definition G.2.*

Proof. Item 1 trivially satisfies Definition G.2.

Consider item 2 of Definition G.16. If such a $\sigma_{\text{echo3},r',v}$ exists, by quorum intersection, all non-faulty parties must have decided $\perp$ or v from the $BCA_{Byz,TSig}$ of that round and set $est = v$. All non-faulty parties start the next round of $AA_{\frac{1}{2}}$ $BCA_{Byz,TSig}$ with v and by validity of $BCA_{Byz,TSig}$, all non-faulty parties start every round $r'' > r'$ (that they start) with v. Hence this condition satisfies Definition G.16.

Finally, consider item 3 of Definition G.16. If such a $\sigma_{\text{echo3},r-1,v}$ exists, then by quorum intersection, no non-faulty party could have output $1 - v$ from the $BCA_{Byz,TSig}$ of that round, and no non-faulty party could have committed $1 - v$. Some non-faulty must have started round $r - 1$ with v, so it must have been externally valid in round $r - 1$. The lemma therefore follows. □

The following two optimizations, when applied to $BCA_{Byz,TSig}$, result in $EVBCA_{Byz,TSig}$.

(1) If party p_i outputs v from the $BCA_{Byz,TSig}$ in round r of $AA_{\frac{1}{2}}$-$BCA_{Byz,TSig}$ and therefore sets $est = val$ in round r, where val is not equal to the value of the coin (line 7 of Algorithm 1), then they directly send a message $\langle$echo2, est, $\sigma_{\text{echo3},r,val}\rangle$ in round $r + 1$, where $\sigma_{\text{echo3},r,val}$ is the output of line 30 in Algorithm 7. In round $r + 1$, other parties verify this echo2 message prior to adding it to their *pendingEcho2* set by verifying that threshold-verify((echo3, r, val), $\sigma_{\text{echo3},r,val}$) = $true$. Upon receiving such an echo2 message, they directly send an echo2 message for val with the same $\sigma_{\text{echo3},r,val}$ in round $r + 1$ if they haven't already sent an echo2 message in that round.

(2) If party p_i decides v in the round r instance of $BCA_{Byz,TSig}$ and v is the value of the coin in round r of $AA_{\frac{1}{2}}$-$BCA_{Byz,TSig}$, it directly sends a designated message to indicate that it is safe for all non-faulty parties to decide v that contains the $\sigma_{\text{echo3},r,v}$ created on Line 30 to all parties. Upon receiving this message and learning the value of the coin in round r, each non-faulty party immediately decides v and forward the message to all other non-faulty parties.

Theorem G.18. *The $EVBCA_{Byz,TSig}$ protocol implements* EVBCA.

Proof. This follows from Lemmas G.20, G.21, G.22, and G.23. □

Before proving that each of the properties of EVBCA are satisfied, we prove a useful lemma:

Lemma G.19. *If $\sigma_{echo3,\ r,\ v}$ exists from round r and value of the common coin of round r is v, and no $\sigma_{echo3,\ r'',\ 1\text{-}v}$ exists from a round $r'' < r$ with $1 - v$ was the value of the coin in round r'', then in all rounds $r' \geq r$, $1 - v$ is not externally valid.*

Proof. If $\sigma_{\text{echo3, r, v}}$ exists from round r and value of the common coin of round r is v, by quorum intersection on the echo3 messages, all non-faulty parties set $est = v$ in round r. Neither of the optimizations apply for value $1 - v$, and all non-faulty parties start round $r + 1$ with value v. By validity, all non-faulty parties start every round $r' > r$, with v and σ_{echo3} for $1 - v$ is never created, thus proving the lemma.

□

We now prove that $EVBCA_{Byz,TSig}$ satisfies all the properties of EVBCA.

Lemma G.20. *The $EVBCA_{Byz,TSig}$ protocol satisfies agreement.*

Proof. We consider two cases:

(1) A non-faulty party decides v in round r upon receiving $2t + 1$ echo3 messages for v.

(2) A non-faulty party decides a value v in round r upon receiving a proof $\sigma_{\text{echo3},r',v}$ from round $r' \leq r$ in which v was the value of the common coin.

In the first case, it must be the case that v is externally valid in round r, so by Lemma G.19 $\sigma_{\text{echo3},r',1-v}$ cannot exist from a round $r' < r$ where $1 - v$ was the value of the coin in round r'. As a result, by quorum intersection, the only value a non-faulty party can decide in the $EVBCA_{Byz,TSig}$ of round r is v or $\perp$.

For the second case, by Lemma G.19, a non-faulty party cannot decide $1 - v$ in round r. □

Lemma G.21. *The $EVBCA_{Byz,TSig}$ protocol satisfies external validity.*

Proof. Assume that v is externally valid and $1 - v$ is not externally valid. Then it must be the case all non-faulty parties start round r of the protocol with v and there is no $\sigma_{\text{echo3},r',1-v}$ from a round $r' < r$ such that threshold-verify$((\text{echo3}, r', 1 - v), \sigma_{\text{echo3},r',1-v})$=*true*, where $r' = r - 1$ or $r' < r$ and v was the value of the common coin in round r'. Since no non-faulty party sends an echo message in round r with $1 - v$, $\sigma_{\text{echo},r,1-v}$ is never created and no non-faulty party adds $(1 - v, *)$ to their *pendingEcho*2 set. Therefore, no non-faulty party ever sends an echo3 message containing $1 - v$ or $\perp$ in round r, and no non-faulty party can decide $1 - v$. The rest of the proof follows from that of Lemma G.22.

□

Lemma G.22. *The $EVBCA_{Byz,TSig}$ protocol satisfies termination.*

Proof. Consider a round r of the protocol. There are 3 cases under which we must prove termination:

(1) There is no $\sigma_{\text{echo3},r,v}$ from round $r - 1$ or from round $r' < r$ where v was the value of the coin.
(2) A party starts round r with $\sigma_{\text{echo3},r-1,v}$ from round $r - 1$ where v was not the value of the coin.
(3) A party starts round r with a $\sigma_{echo,r',v}$ from round $r' < r$ where the value of the coin was v.

Termination for case 1 follows from Lemma F.3.

In case 2, the party that has the $\sigma_{echo3,r-1,v}$ directly sends in round r $\langle \text{echo2}, est, \sigma_{\text{echo3},r,val} \rangle$, where $\sigma_{\text{echo3},r,val}$ is the output of line 30 from round $r - 1$ of the protocol. Note that this party could be Byzantine and therefore send this to only some of the non-faulty parties. Every non-faulty party that receives this echo2 message and sends $\langle \text{echo2}, est, \sigma_{\text{echo3},r-1,v} \rangle$ if they haven't already sent an echo2 message. It is left to prove that all non-faulty parties that did not receive this message send echo2 messages. This follows from the fact that there must be some value such that at least $t + 1$ non-faulty parties started with this value. As a result, every non-faulty party's *pendingEcho*2 size becomes of size at least $2t + 1$ and each non-faulty party sends an echo3 message that satisfies the conditions on Lines 20 and 23. Every non-faulty party eventually has a *pendingEcho*3 set of size $2t + 1$ and decides a value.

In case 3, some party has a proof, $\sigma_{\text{echo3},r',v}$ from round $r' < r$ in which the common coin equalled v, and therefore directly sends a designated message containing $\sigma_{\text{echo3},r',v}$ to other parties. All non-faulty parties that receive this message decide v upon receiving this message and learning the value of the coin in round r'; in addition, they forward this message to all the other parties, so all non-faulty parties decide. □

Lemma G.23. *The $EVBCA_{Byz,TSig}$ protocol satisfies binding.*

Proof. For the purposes of this proof, we refer to a value v as "permanently externally valid" if $\sigma_{echo3,r',v}$ exists from a round r' where v was the value of the common coin. A non-faulty party decides a value v in round r either after receiving proof that v is permanently externally valid or after receiving echo3 messages from $2t + 1$ parties in round r.

We therefore consider two cases for a round r:

(1) There is a permanently externally valid value v with a proof from a round $r' < r$.
(2) There is no permanently externally valid value v from a round $r' < r$.

By Definition G.16, $1-v$ cannot be decided by a non-faulty party in round r, since that would make $1-v$ externally valid, violating Lemma G.19.

In case 2, a non-faulty party decides only after receiving echo3 messages from $2t+1$ parties. Binding follows from Lemma F.5. □

Note that it is possible that a Byzantine party has a proof $\sigma_{echo3,r,v}$ from round r in which v was the value of the coin, and no non-faulty party has the proof. In this case, all non-faulty parties would have output v or $\perp$ from $EVBCA_{Byz,TSig}$ of round r, and therefore would adopt v as their value. If the Byzantine party ever sends $\sigma_{echo3,r,v}$ to a non-faulty party, validity ensures that it will be safe for that non-faulty party to decide v and forward the proof to all of the other parties.

We now prove two lemmas regarding the expected constant round termination of the $AA_{\frac{1}{2}}$-$BCA_{Byz,TSig}$ protocol.

Lemma G.24. *If all non-faulty parties start round r of the $AA_{\frac{1}{2}}$-$BCA_{Byz,TSig}$ protocol with value v, and $1-v$ is not externally valid, then all non-faulty parties commit in a constant number of rounds.*

Proof. By external validity, all non-faulty parties output v and $\sigma_{echo3,r,v}$ is created. This happens in every round and non-faulty parties terminate in the round in which the coin equals v. In expectation, this happens in 2 rounds. □

Lemma G.25. *The $AA_{\frac{1}{2}}$-$EVBCA_{Byz,TSig}$ protocol with a $2t$ degree coin takes, in expectation, 9 rounds of broadcast until all non-faulty parties terminate against an adaptive adversary.*

Proof. Note that if in any round r, there is an echo3 proof, $\sigma_{echo3,r,v}$, created, and the coin is equal to v, or there is no $\sigma_{echo3,r,v}$ created, Lemma G.24 applies in the next round. It takes, in expectation, 2 rounds for the coin value to be equal to the value in the echo3 proof, at which point the non-faulty party that knows of echo3 proof commits. If the adversary chooses to have no such echo3 proof (or not let the non-faulty parties be aware of it), then all non-faulty parties output $\perp$, adopt the coin value, and commit in an expected two additional rounds. There are at most 3 broadcasts in any round of the $AA_{\frac{1}{2}}$-$EVBCA_{Byz,TSig}$ protocol. Once all non-faulty parties start a round with the same value and the echo3 proof is created in that round, there are at most 2 broadcasts per round in subsequent rounds. Therefore, it takes in expectation at most 3+3+2 broadcasts until the first non-faulty party commits and an additional 1 broadcast to ensure all non-faulty parties commit, for a total of 9 broadcasts in expectation. □