

8 Agreement and Broadcasts with Byzantine Faults

8.1 Model

Faults. In the context of malicious activity, one usually models the environment as an *adversary* that coordinates all faulty servers. An adversary may be *passive* or *honest-but-curious*, in the sense that the faulty servers follow the protocol but the adversary tries to infer information it is not supposed to know. Sometimes, a passive adversary may additionally cause servers to crash. A completely unrestricted adversary is called *active* or *Byzantine*, and may cause faulty servers to deviate from their specification and to behave arbitrarily.

Components. In this section, there are n servers, of which up to t may be *corrupted* by an active adversary; the remaining servers are *correct*. The servers are connected by pairwise reliable and authenticated links, and the system is asynchronous (no bounds on message delays, no local clocks). There is a trusted dealer for setup and the adversary is computationally bounded.

Assumptions. Cryptography, in particular threshold cryptography, is used here to cope with failures and adversarial actions. Keys and key shares for threshold cryptography are initially distributed by a trusted dealer. Randomized protocols may be used to reach agreement, since deterministic asynchronous consensus and agreement protocols have infinite runs.

8.2 Broadcasts

In *consistent* and *reliable* broadcasts, a distinguished sender P_s *broadcasts* a message m and all servers should later *deliver* m , though termination is not guaranteed when the sender is faulty.

Consistent broadcast. *Consistent broadcast* or *c-broadcast* ensures only that the delivered message is the same for all receivers. In particular, when the sender is faulty, it does not guarantee that *every* server delivers a message and terminates.

Definition 1 (Consistent broadcast). A protocol for consistent broadcast satisfies:

Validity: If a correct sender P_s *c-broadcasts* m , then all correct servers eventually *c-deliver* m .

Consistency: If some correct server *c-delivers* m and a distinct correct server *c-delivers* m' , then $m = m'$.

Integrity: For any request m , every correct server *c-delivers* m at most once. Moreover, if the sender P_s is correct, then m was previously *c-broadcast* by P_s .

The *echo broadcast* protocol implements consistent broadcast with a *linear* number of messages and uses digital signatures. Its idea is that the sender distributes the request to all servers and expects $\lceil \frac{n+t+1}{2} \rceil$ servers to act as witnesses for the request; they attest this by signing their reply to the sender.

Algorithm 2 (Echo broadcast [Rei94]). Every server can issue digital signatures and verify the signatures of every other server.

```

upon  $c$ -broadcast( $m$ ):                                     // sender only
    send message (send,  $m$ ) to all
upon receiving a message (send,  $m$ ) from  $P_s$ :
    compute signature  $\sigma$  on (echo,  $s$ ,  $m$ )
    send message (echo,  $m$ ,  $\sigma$ ) to  $P_s$ 
upon receiving  $\lceil \frac{n+t+1}{2} \rceil$  messages (echo,  $m$ ,  $\sigma_i$ ) with valid  $\sigma_i$ :       // sender only
     $\Sigma \leftarrow$  list of all received signatures  $\sigma_i$ 
    send message (final,  $m$ ,  $\Sigma$ ) to all
upon receiving a message (final,  $m$ ,  $\Sigma$ ) from  $P_s$  with  $\lceil \frac{n+t+1}{2} \rceil$  valid signatures in  $\Sigma$ :
     $c$ -deliver( $m$ )

```

The message complexity of the echo broadcast protocol is $O(n)$ and its communication complexity is $O(n^2(k + |m|))$, where k denotes the length of a digital signature. Using a non-interactive threshold signature scheme, the communication complexity can be reduced to $O(n(k + |m|))$.

Theorem 3. Algorithm 2 implements consistent broadcast with Byzantine faults for $n > 3t$.

Proof. Validity and integrity are straightforward to verify. Consistency follows from the observation that the request m in any final message with $\lceil \frac{n+t+1}{2} \rceil$ valid signatures in Σ is unique. To see this, consider the set of servers that issued the $\lceil \frac{n+t+1}{2} \rceil$ signatures: because there are only n distinct servers, every two sets of signers overlap in at least one correct server. Such sets are called *Byzantine quorums* [MR98]. \square

Reliable broadcast. *Reliable broadcast* or *r-broadcast* extends consistent broadcast and ensures additionally agreement on the delivery of a message, in the sense that either all correct servers deliver some request or none delivers any request.

Definition 4 (Reliable broadcast or “Byzantine generals problem”). A protocol for reliable broadcast is a consistent broadcast protocol that satisfies also:

Totality: If some correct server r -delivers a message, then all correct servers eventually r -deliver a message.

In the literature *consistency* and *totality* are often combined into a single condition called *agreement*.

Algorithm 5 (Bracha broadcast [Bra84]). This protocol needs only pairwise authenticated links, but no digital signatures.

```

upon  $r$ -broadcast( $m$ ):                                     // sender only
    send message (send,  $m$ ) to all
upon receiving a message (send,  $m$ ) from  $P_s$ :

```

send message (echo, m) to all
upon receiving $\lceil \frac{n+t+1}{2} \rceil$ messages (echo, m) and not having sent (ready, m) :
 send message (ready, m) to all
upon receiving $t+1$ messages (ready, m) and not having sent (ready, m) :
 send message (ready, m) to all
upon receiving $2t + 1$ messages (ready, m) :
 $r\text{-deliver}(m)$

Theorem 6. *Algorithm 5 implements reliable broadcast with Byzantine faults for $n > 3t$.*

Proof. Consistency follows from the same argument as in Theorem 3, since the message m in any `ready` message of a correct server is unique. Totality is implied by the “amplification” of `ready` messages from $t+1$ to $2t+1$ with the fourth **upon** clause of the algorithm. Specifically, if a correct server has $r\text{-delivered}$ m , it has received a `ready` message with m from $2t + 1$ distinct servers. Therefore, at least $t + 1$ correct servers have sent a `ready` message with m , which will be received by all correct servers and cause them to send a `ready` message as well. Because $n - t \geq 2t + 1$, all correct servers eventually receive enough `ready` messages to terminate. \square

8.3 Randomized [Binary] Byzantine Agreement

The *Byzantine agreement* problem is characterized by two events *propose* and *decide*; every server executes $\text{propose}(v)$ to start the protocol and $\text{decide}(v)$ when the protocol terminates with a decision value v . In *binary* agreement, the values are bits. Note that *Byzantine agreement* in the model with Byzantine failures is almost the same as consensus subject to crash failures.

Definition 7 (Byzantine agreement). A protocol for binary Byzantine Agreement satisfies:

Validity: If all correct servers *propose* v , then some correct server eventually *decides* v .

Agreement: If some correct server *decides* v and another correct server *decides* v' , then $v = v'$.

Termination: Every correct server eventually *decides*.

The result of Fischer, Lynch, and Paterson [FLP85] implies that every asynchronous protocol solving Byzantine agreement has executions that do not terminate. However, there are randomized protocols that circumvent this impossibility [Ben83, Rab83, Tou84]. They make the probability of non-terminating executions arbitrarily small. More precisely, given a logical time measure T , such as the number of steps performed by all correct servers, define *termination with probability 1* as

$$\lim_{T \rightarrow \infty} \Pr[\text{some correct server has not } \textit{decided} \text{ after time } T] = 0.$$

Algorithm 8 (Binary randomized Byzantine agreement [Tou84]). Suppose a trusted dealer has *shared* a sequence s_0, s_1, \dots of random bits, called *coins*, among the servers, using $(t+1)$ -out-of- n secret sharing. A server can access the coin s_r using a *recover*(r) operation, which may involve a protocol that exchanges some messages, and gives the same coin value to every server. The two *upon* clauses of the algorithm are executed concurrently.

At a high level, the protocol works as follows. Every server maintains a value v , called its *vote*, and the protocol proceeds in global asynchronous rounds. Every round consists of two voting steps among the servers with all-to-all communication. In the first voting step, the servers simply exchange their votes, and every server determines the majority of the received votes. In the second voting step, every server relays the majority vote to all others, this time using reliable broadcast (Algorithm 5) and accompanied by a *proof* Π that justifies the choice of the majority; Π contains messages and signatures from the first voting step. After receiving reliable broadcasts from $n - t$ servers, every server determines the majority of this second vote and adopts its outcome as its vote v if the result is unanimous; otherwise, a server sets v to the shared coin for the round. If the coin equals the outcome of the second vote, then the server decides.

upon *propose*(v):

$r \leftarrow 0$

loop

send the signed message $(1\text{-vote}, r, v)$ to all

wait for receiving properly signed $(1\text{-vote}, r, v')$ messages from $n - t$ distinct servers

$\Pi \leftarrow$ set of received 1-vote messages including the signatures

$v \leftarrow$ value v' that is contained most often in Π

r-broadcast message $(2\text{-vote}, r, v, \Pi)$

wait for *r-delivery* of $(2\text{-vote}, r, v', \Pi)$ messages from $n - t$ distinct senders

with valid signatures in Π and correctly computed v'

$b \leftarrow$ value v' that is contained most often among the *r-delivered* 2-vote messages

$c \leftarrow$ number of *r-delivered* 2-vote messages with $v' = b$

$s_r \leftarrow \text{recover}(r)$

if $c = n - t$ **then**

$v \leftarrow b$

else

$v \leftarrow s_r$

if $b = s_r$ **then**

send message (decide, v) to all

// note that $v = s_r = b$

$r \leftarrow r + 1$

upon receiving $t + 1$ messages (decide, b) :

if not *decided* **then**

send message (decide, b) to all

decide(b)

Lemma 9. *If all correct servers start some round r with vote v_0 , then all correct servers terminate round r with vote v_0 .*

Proof. It is impossible to create a valid Π for a 2-vote message with a vote $v \neq v_0$ because v must be set to the majority value in $n - t$ received 1-vote messages and $n - t > 2t$. \square

Lemma 10. *In round $r \geq 0$, the following holds:*

1. *If a correct server sends a `decide` message for v_0 at the end of round r , then all correct servers terminate round r with vote v_0 .*
2. *With probability at least $\frac{1}{2}$, all correct servers terminate round r with the same vote.*

Proof. Consider the assignment of b and c in round r . If some correct server obtains $c = n - t$ and $b = v_0$, then no correct server can obtain a majority of `2-vote` messages for a value different from v_0 (there are only n `2-vote` messages and they satisfy the consistency of reliable broadcast). Those correct servers with $c = n - t$ set vote v to v_0 ; every other correct server sets v to s_r . Hence, if $s_r = v_0$, all correct servers terminate round r with vote v_0 .

Claim *a*) now follows upon noticing that a correct server only sends a `decide` message for v_0 when $v_0 = b = s_r$.

Claim *b*) follows because the first correct server to assign b and c does so *before* any information about s_r is known (to the adversary). To see this note that at least $t + 1$ shares are needed for recovering s_r , but a correct server only reveals its share *after* assigning b and c . Thus, s_r and v_0 are statistically independent and $s_r = v_0$ holds with probability $\frac{1}{2}$. \square

Theorem 11. *Algorithm 8 implements binary Byzantine agreement for $n > 3t$, where termination holds with probability 1.*

The theorem follows easily from the two lemmas. The protocol achieves optimal resilience because reaching agreement in asynchronous networks with $t \geq n/3$ Byzantine faults is impossible, despite the use of digital signatures [Tou84]. Since Algorithm 8 reaches agreement with probability at least $\frac{1}{2}$ in every round, the expected number of rounds is two, and the expected number of messages is $O(n^3)$.

8.4 Byzantine Agreement with Cryptographic Randomness

One problem of Algorithm 8 is that every round in the execution uses up one shared coin in the sequence s_0, s_1, \dots . As coins cannot be reused, this is a problem. One solution is to obtain the shared coins from a threshold-cryptographic function.

More precisely, one may obtain the coin value s_r from the output of a distributed *pseudo-random function (PRF)* evaluated on the round number r and the protocol instance identifier.

A PRF $F_x : \{0, 1\}^* \rightarrow \{0, 1\}^k$ is parameterized by a secret key x (called a *seed*) and maps an input string to a k -bit output string that looks random to anyone who does not have the secret key. More formally, the PRF is secure if any efficient adversary who queries an oracle with distinct inputs cannot tell, with better than negligible probability, whether the oracle responds to the queries by evaluating the PRF on a random seed or whether the oracle responds with a uniformly random k -bit string.

In practice one implements a PRF by a block cipher with a secret key; distributed implementations, however, are only known for functions based on public-key cryptosystems. Cachin et al. [CKS05] describe a distributed PRF suitable for a distributed implementation, as shown next.

Algorithm 12 ([CKS05]). Recall the discrete-logarithm setting with $G = \langle g \rangle$; let x be a randomly chosen *seed* and define a $F_x : \{0, 1\}^* \rightarrow \{0, 1\}^k$ as

$$F_x(v) = H'(H(v)^x),$$

where $H : \{0, 1\}^* \rightarrow G$ and $H' : G \rightarrow \{0, 1\}^k$ are two hash functions. The family $F = \{F_x\}$ is a pseudorandom function, assuming the hardness of the DLP (which can be proven formally when H is modeled as a so-called random oracle).

A *threshold PRF* can be obtained analogously to threshold ElGamal encryption. Let a trusted dealer choose the seed x and share it among the servers using a polynomial of degree t , such that server P_i holds share x_i . When it is time to compute $F_x(v)$, every correct server P_i computes a value-share $d_i = (H(v))^{x_i}$ and releases d_i according to the protocol. Any $t + 1$ correctly computed value-shares, from servers with indices in a set S , yield the value of the PRF,

$$F_x(v) = H'\left(\prod_{i \in S} d_i^{\lambda_{0,i}^S}\right).$$

Writing $h = H(v)$, this is correct because

$$\prod_{i \in S} d_i^{\lambda_{0,i}^S} = \prod_{i \in S} h^{x_i \lambda_{0,i}^S} = h^{\sum_{i \in S} x_i \lambda_{0,i}^S} = h^x.$$

One can show that this does not leak information about x , under the assumption that the DLP is hard.

Now use the threshold PRF with output length $k = 1$ to implement the sequence of shared coins s_0, s_1, \dots in Algorithm 8. The input string for coin s_r consists of $ID \| r$, where ID denotes a unique *tag* of the protocol instance that must also be contained in every message and included in all signatures, and where r denotes the index of the coin.

Since the threshold pseudorandom function is non-interactive, the *recover* operation for s_r can be implemented simply by having every server send its share of s_r to all others, collecting $t + 1$ shares, and combining them to the coin value $F_x(ID \| r)$.

The threshold PRF as described here tolerates only a passive adversary, but it can easily be made robust against an active adversary by adding the appropriate zero-knowledge proofs [CKS05].

The randomized Byzantine agreement protocol of Algorithm 12 based on threshold cryptography has expected message complexity $O(n^3)$. It can be reduced to $O(n^2)$ expected messages by replacing the reliable broadcasts with a two-phase voting structure that uses *justifications* for each vote obtained through threshold signatures on earlier votes [CKS05].

8.5 Atomic Broadcast with Byzantine Faults

Given a Byzantine agreement protocol, an asynchronous atomic broadcast protocol that tolerates Byzantine faults can be realized using the same approach as with crash failures: proceed in global asynchronous rounds and for each round, agree on a batch of messages to be delivered at the end of the round [CKPS01].

Other approaches are possible and sometimes advantageous. A popular Byzantine-fault tolerant (BFT) algorithm suitable for state-machine replication has been described by Castro and Liskov [CL02]. It builds on a different paradigm than the round-based structure of Algorithm 12 and uses a deterministic protocol subject to a timing assumption. Since it needs essentially only one instance of reliable broadcast (Algorithm 5) per atomically delivered message, it is generally faster than Algorithm 12.

References

- [Ben83] M. Ben-Or, *Another advantage of free choice: Completely asynchronous agreement protocols*, Proc. 2nd ACM Symposium on Principles of Distributed Computing (PODC), 1983, pp. 27–30.
- [Bra84] G. Bracha, *An asynchronous $[(n - 1)/3]$ -resilient consensus protocol*, Proc. 3rd ACM Symposium on Principles of Distributed Computing (PODC), 1984, pp. 154–162.
- [CKPS01] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup, *Secure and efficient asynchronous broadcast protocols (extended abstract)*, Advances in Cryptology: CRYPTO 2001 (J. Kilian, ed.), Lecture Notes in Computer Science, vol. 2139, Springer, 2001, pp. 524–541.
- [CKS05] C. Cachin, K. Kursawe, and V. Shoup, *Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography*, Journal of Cryptology **18** (2005), no. 3, 219–246.
- [CL02] M. Castro and B. Liskov, *Practical Byzantine fault tolerance and proactive recovery*, ACM Transactions on Computer Systems **20** (2002), no. 4, 398–461.
- [FLP85] M. J. Fischer, N. A. Lynch, and M. S. Paterson, *Impossibility of distributed consensus with one faulty process*, Journal of the ACM **32** (1985), no. 2, 374–382.
- [MR98] D. Malkhi and M. K. Reiter, *Byzantine quorum systems*, Distributed Computing **11** (1998), no. 4, 203–213.
- [Rab83] M. O. Rabin, *Randomized Byzantine generals*, Proc. 24th IEEE Symposium on Foundations of Computer Science (FOCS), 1983, pp. 403–409.
- [Rei94] M. K. Reiter, *Secure agreement protocols: Reliable and atomic group multicast in Rampart*, Proc. 2nd ACM Conference on Computer and Communications Security (CCS), 1994, pp. 68–80.
- [Tou84] S. Toueg, *Randomized Byzantine agreements*, Proc. 3rd ACM Symposium on Principles of Distributed Computing (PODC), 1984, pp. 163–178.