

## 8 Agreement and Broadcasts with Byzantine Faults

### 8.1 Beyond Crash Failures

Malicious activity can be interpreted as a severe kind of failure; this idea leads to a refined classification of faults as follows.

#### A. Random, non-adversarial faults

- 1) fail-stop [servers may crash, but crashes can be detected remotely; as with a perfect failure detector]
- 2) crash(-silent) [servers may crash, not detectable remotely]
- 3) crash-recovery [servers may crash and recover afterwards; using stable storage]
- 4) omission [servers may omit receiving, processing, or sending of messages]

#### B. Adversarial and malicious faults

- 1) passive adversary [no deviation from specification, honest-but-curious adversary]
- 2) passive adversary with crashes [honest-but-curious adversary that may cause servers to crash, but not otherwise deviate from the specification]
- 3) active or Byzantine adversary [arbitrary behavior]

This classification is intended for asynchronous distributed systems, but most concepts apply also in other contexts. So far, we have mostly used **A2** crash failures; in Chapter 7 (Distributed Cryptography) we address **B1** and in this chapter, we mainly address **B2**; both fault models require cryptographic techniques. Note that this classification is not one-dimensional, although there are some strict inclusion relations, like **A1-A2-B2-B3** and **A2-A4-B3**.

Additional, orthogonal assumptions for **B** (adversarial and malicious faults) distinguish the system setup (either using a trusted dealer for generation and initial distribution of keys, or no such dealer), and the adversary power (either computationally bounded (i.e., with running time bounded by a polynomial in a security parameter) or unbounded (resulting in information-theoretic security)).

In the context of adversarial faults, the correct, i.e., non-faulty, servers are sometimes also called *honest* and faulty servers are typically called *corrupted*, to emphasize that a malicious adversary may be coordinating the faulty parties.

### 8.2 Model

**Components.** There are  $n$  servers, of which up to  $t$  may be *corrupted* by an *adversary* and exhibit arbitrary faults; the remaining servers are *correct*. The servers 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.

**Methods.** Cryptography, in particular, threshold cryptography (signatures and pseudorandom generators), is used to cope with potentially malicious failures. Keys and key shares (for threshold cryptography) are initially distributed by a trusted dealer. Randomized protocols are used to reach agreement, since deterministic asynchronous consensus and agreement protocols have infinite runs. Such randomized protocols achieve agreement in finite time with all but negligible probability.

### 8.3 Broadcasts

Broadcasts are parameterized by a tag  $ID$ , which is contained (implicitly) in every message and must be included in signatures. In *consistent* and *reliable* broadcasts, a distinguished sender  $P_s$  *broadcasts* a message  $m$  and all servers (should) *deliver*  $m$ .

**Consistent broadcast.** Consistent broadcast (“c-broadcast”) ensures only that the delivered message is consistent for all receivers. In particular, termination is not guaranteed with a faulty sender.

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

*Validity:* If an correct sender  $P_s$  *c-broadcasts*  $m$ , then  $P_s$  eventually *c-delivers*  $m$ .

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

*Integrity:* Every correct server *c-delivers* at most one  $m$ .

*Termination:* If the sender is correct, then all correct servers eventually *c-deliver* a message.

**Algorithm 2 (Echo broadcast using digital signatures).** Assume every server can digitally sign messages, which can be verified by any server.

```

upon c-broadcast( $m$ ):                                // sender  $P_s$  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  $P_s$  only
    let  $\Sigma$  be the 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.** Assuming perfectly unforgeable signatures, Algorithm 2 implements consistent broadcast with Byzantine faults for  $n > 3t$ .

*Proof.* The message  $m$  in any final message with enough valid signatures in  $\Sigma$  is unique because  $\lceil \frac{n+t+1}{2} \rceil$  signatures are a quorum in the Byzantine failure model.  $\square$

**Reliable broadcast.** Reliable broadcast (“r-broadcast”) ensures additionally agreement on the delivery of a message.

**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.

Totality ensures that all correct servers either deliver a message or don’t. In the literature *consistency* and *totality* are often combined into a single condition called *agreement*.

**Algorithm 5 (Bracha broadcast).** This protocol does not need digital signatures.

```

upon  $r$ -broadcast( $m$ ):                                // sender  $P_s$  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,  $m$ ):
     $r$ -deliver( $m$ )

```

**Theorem 6 ([Bra84]).** 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  $2t + 1$ .  $\square$

## 8.4 Randomized [Binary] Byzantine Agreement

Binary Byzantine agreement is characterized by two events *propose* and *decide*; every server executes *propose*( $b$ ) to start the protocol and *decide*( $b$ ) to terminate it, for a bit  $b$ .

**Definition 7 (Binary 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*.

It is not possible to implement Definition 7 in asynchronous systems [FLP85]. But one can relax either *termination* or *agreement* to hold with high probability, and there are protocols that satisfy them with probability 1 after infinite running time. More precisely, given a logical time measure  $T$ , such as the number of steps performed by all correct servers, *termination with probability 1* means that

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

**Algorithm 8 ([Tou84]).** Suppose a trusted dealer has *shared* a sequence  $s_0, s_1, \dots$  of random bits or “coins” among the servers (using  $(n - t)$ -out-of- $n$  secret sharing as in Chapter 7). The secrets can be accessed using a *recover* operation (this will involve exchanging some messages) [Rab83]. The two *upon* clauses of the algorithm below are executed in parallel threads.

The value  $v$  is called the “vote”; the set  $\Pi$  is a “proof” that justifies the choice of  $v$  in the 2-vote message; a “round” of the algorithm consists of two rounds of message exchanges.

**upon** *propose*( $v$ ):

$r \leftarrow 0$

**loop**

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

receive 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  $\Pi$

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

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

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

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

*recover*( $s_r$ )

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

$v \leftarrow v''$

**else**

$v \leftarrow s_r$

**if**  $v'' = s_r$  **then**

send the message  $(\textit{decide}, v)$  to all

// note that  $v = s_r = v''$

$r \leftarrow r + 1$

**upon** *receiving*  $t + 1$  messages  $(\textit{decide}, b)$ :

**if** not *decided* **then**

send the message  $(\textit{decide}, b)$  to all

*decide*( $b$ )

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

*Proof.* It is impossible to create a valid  $\Pi$  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 every round  $r$ , the following holds:*

a) *If a correct server sends a decide message for  $v_0$  at the end of round  $r$ , then all correct servers will terminate round  $r$  with vote  $v_0$ .*

b) *With probability at least  $1/2$ , all correct servers will terminate round  $r$  with the same vote.*

*Proof.* Consider the assignment of  $v''$  and  $c$  in round  $r$ . If some correct server obtains  $c = n - t$  and  $v'' = v_0$ , then no correct server obtains  $c = n - t$  but  $v'' \neq v_0$ . All correct servers with  $c = n - t$  set  $v$  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  $v$  when  $v'' = s_r$ .

Claim b) follows because the first correct server to assign  $v''$  and  $c$  does so *before* anything about  $s_r$  is known (to the adversary),  $s_r$  and  $v_0$  are independent; hence,  $s_r = v_0$  with probability  $1/2$ .  $\square$

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

Since Algorithm 8 reaches agreement with probability at least  $1/2$  in every round, the expected number of rounds is 2. The expected number of messages sent is  $O(n^3)$ .

## 8.5 Byzantine Agreement using Cryptographic Randomness

Algorithm 8 assumes a sequence of unpredictable, shared random coins  $s_0, s_1, \dots$ . When implemented using secret sharing, every instance of agreement needs a fresh sequence of shared bits, which is infeasible in practice. More practical implementations can be obtained using threshold cryptography as follows.

**Algorithm 12 ([CKS05]).** In Algorithm 8, implement the sequence of shared coins  $s_0, s_1, \dots$  by the Diffie-Hellman-based threshold pseudorandom function from Exercise 7.2, using a polynomial of degree  $n - t - 1$  to share the secret key.

Recall that the pseudorandom function  $F_x(s)$  uses a secret seed  $x$ , takes an arbitrary input string  $s$ , and maps the input to a pseudorandom value in  $\{0, 1\}^k$ . We set  $k = 1$  and let the input string for coin  $s_j$  consist of  $ID||j$ , i.e., the tag  $ID$  of the protocol instance concatenated with the coin counter  $j$ .

The threshold pseudorandom function is initialized by the trusted dealer who chooses the seed  $x$  and shares it among the servers. It can be accessed and used by a practically unlimited number of concurrent protocol instances.

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

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

## 8.6 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 (Algorithm 20 from Chapter 5): proceed in global asynchronous rounds and for each round, agree on a batch of messages to be delivered at the end of the round. Such an algorithm is described in [CKPS01].

### References

- [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.
- [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.
- [Rab83] M. O. Rabin, *Randomized Byzantine generals*, Proc. 24th IEEE Symposium on Foundations of Computer Science (FOCS), 1983, pp. 403–409.
- [Tou84] S. Toueg, *Randomized Byzantine agreements*, Proc. 3rd ACM Symposium on Principles of Distributed Computing (PODC), 1984, pp. 163–178.