Consensus-related definitions

Andreea Alexandru

July 2021

Contents

1 Definitions

Definition 1 (Binary Byzantine Agreement (BA) [BKL19])**.** Let Π be a protocol executed by n parties $P_1, \ldots, P_n$, where each party P_i begins holding input $v_i \in \{0, 1\}$. The following notions hold whenever there are at most t of the parties are corrupted.

- **Weak validity**: Π is *t-weakly valid* if the following holds: if every honest party's input is equal to the same value v, then every honest party outputs either v or $\perp$.
- **Validity**: Π is *t-valid* if the following holds: if every honest party's input is equal to the same value v, then every honest party outputs v.
- **Validity with termination**: Π is *t-valid with termination* if the following holds: if every honest party's input is equal to the same value v, then every honest party outputs v and terminates.
- **Weak consistency (consistency with abort)**: Π is *t-weakly consistent* if there is a $v \in \{0, 1\}$ such that every honest party outputs either v or $\perp$.
- **Consistency (consistency with unanimous abort)**: Π is *t-consistent* if there is a $v \in \{0, 1, \perp\}$ such that every honest party outputs v.
- **Liveness**: Π is *t-live* if every honest party outputs a value in $\{0, 1\}$.
- **Termination**: Π is *t-terminating* if every honest party terminates. Π has *guarantees termination* if it is n-terminating.

If Π is t-valid, t-consistent, t-live and t-terminating, we say that Π is t-secure.

BA only makes sense for $t < n/2$.

Given a PKI, in a synchronous network BA can be achieved if and only if $t_s < n/2$ parties are corrupted and in an asynchronous network if there are $t_a < n/3$ corruptions. Protocols tolerating the optimal threshold and running in expected constant rounds are known [CR93, KK09]. BA can be achieved with t_s corruptions when run in synchronous mode and t_a corruptions when run in asynchronous mode if and only if $2t_s + t_a < n$ [BKL19].

Suppose there is a global polynomial-time computable predicate Q_{ID} known to all parties, which is determined by an external application. Each party may propose a value v together with a proof π that should satisfy Q_{ID}. Definition 2 holds for non-binary message domain.

Definition 2 (Validated Asynchronous Byzantine Agreement (VABA) [CKPS01])**.** A protocol solves *validated Byzantine agreement* with predicate Q_{ID} if it satisfies the following conditions except with negligible probability:

- **External validity**: Any honest party that terminates for ID decides v validated by π such that $Q_{ID}(v, \pi)$ holds.
- **Agreement/Consistency**: If some honest party decides v for ID, then any honest party that terminates decides v for ID.
- **Liveness/Termination**: If all honest parties have been activated on ID and all associated messages have been delivered, then all honest parties have decided for ID.
- **Integrity**: If all parties follow the protocol, and if some party decides v validated by π for ID, then some party proposed v validated by π for ID.
- **Efficiency**: For every ID, the communication complexity for ID is probabilistically uniformly bounded.
- **Quality** [AMS19]: The probability of choosing a value that was proposed by an honest party is at least $1/2$.

Validated Byzantine Agreement generalizes the primitive of agreement on a core set (see Definitions 8, 9). A binary validated Byzantine agreement protocol biased towards 1 is a protocol for validated Byzantine agreement on values in $\{0, 1\}$ such that the following condition holds:

Biased External Validity: If at least $t + 1$ honest parties propose 1, then any honest party that terminates for ID decides 1.

Definition 3 (Broadcast [BKL19])**.** Let Π be a protocol executed by parties $P_1, P_2, \ldots, P_n$, where a sender $P^* \in \{P_1, \ldots, P_n\}$ begins holding input $v^* \in \{0, 1\}$ are all parties are guaranteed to terminate. The following notions hold whenever there are at most t of the parties are corrupted.

- **Weak validity**: Π is *t-weakly valid* if the following holds: if P^* is honest, then every honest party outputs either v^* or $\perp$.
- **Validity**: Π is *t-valid* if the following holds: if P^* is honest, then every honest party outputs v^*.
- **Weak consistency**: Π is *t-weakly consistent* if there is a $v \in \{0, 1\}$ such that every honest party otputs either v or $\perp$.
- **Consistency**: Π is *t-consistent* if there is a $v \in \{0, 1, \perp\}$ such that every honest party outputs v.
- **Liveness**: Π is *t-live* if every honest party outputs a value in $\{0, 1\}$.

If Π is t-valid, t-consistent, t-live and t-terminating, we say that Π is t-secure.

Authenticated broadcast (assuming a PKI) is possible for $t < n$; otherwise, it is only possible for $t < n/3$.

In the synchronous case, for $t < n/2$, BA implies broadcast (using one additional round: in the first round the dealer sends its input to all players, who then run BA on the values they received). For $t < n/2$, a broadcast channel implies BA (each player broadcasts their value and then honest players decide on the majority value).

Definition 4 (Reliable broadcast [BKL20])**.** Let Π be a protocol executed by parties $P_1, P_2, \ldots, P_n$, where a sender $P^* \in \{P_1, \ldots, P_n\}$ begins holding input $v^* \in \{0, 1\}$ are all parties are guaranteed to terminate. The following notions hold whenever there are at most t of the parties are corrupted.

- **Validity**: Π is *t-valid* if the following holds: if P^* is honest, then every honest party outputs v^*.

- **Consistency**: Π is *t-consistent* if the following holds: either no honest party outputs anything or all honest parties output the same value $v \in \{0,1\}$.
- **Totality** [CKPS01]: If some honest party delivers a message tagged with $ID.j.s$, then all honest parties deliver some message tagged with $ID.j.s$, provided all honest parties have been activated on $ID.j.s$ and the adversary delivers all associated messages.
- **Integrity** [CKPS01]: For all ID, senders j, and sequence numbers s, every honest party delivers at most one message v tagged with $ID.j.s$. Moreover, if all parties follow the protocol, then v was previously reliably broadcast by P_j with sequence number s.

If Π is t-valid and t-consistent, we say that Π is t-secure.

The difference between Definitions 3 and 4 is that, when the sender is dishonest, reliable broadcast allows that no honest party terminates.

Definition 5 (Verifiable broadcast [CKPS01])**.** A broadcast protocol is called *verifiable* if the following holds, except with negligible probability: when an honest party has delivered payload m tagged with ID, then it can produce a single protocol message M that it may send to other parties such that any other honest party will deliver m tagged with ID upon receiving M (provided the other party has not already delivered m).

We call M the message that completes the verifiable broadcast. This notion implies that there is a predicate V_{ID} that the receiving party can apply to an arbitrary bit string for checking if it constitutes a message that completes a verifiable broadcast tagged with ID.

Definition 6 (Consistent broadcast [CKPS01])**.** A protocol for *consistent broadcast* is a protocol for reliable broadcast that does not necessarily satisfy totality. In other words, consistent broadcast makes no provisions that two parties do deliver the payload message, but maintains consistency among the actually delivered messages with the same senders and sequence numbers.

Definition 7 (Graded consensus [BKL19])**.** Let Π be a protocol executed by n parties $P_1, \ldots, P_n$, where each party P_i begins holding input $v_i \in \{0,1\}$, and each party terminates upon generating output. The following notions hold whenever there are at most t of the parties are corrupted.

- **Graded validity**: Π achieves *t-graded validity* if the following holds: if every honest party's input is equal to the same value v, then every honest party outputs $(v, 2)$.
- **Graded consistency**: Π achieves *t-graded consistency* if the following hold: (1) if two honest parties output grades g, g', then $|g - g'| \leq 1$ and (2) if two honest parties output (v, g) and (v', g') with $g, g' \geq 1$, then $v = v'$.
- **Liveness**: Π is *t-live* if every honest party outputs (v, g) with either a $v \in \{0,1\}$ and $g \geq 1$ or $v = \perp$ and $g = 0$.
- **Termination**: Π is *t-terminating* if every honest party terminates. Π has *guarantees termination* if it is n-terminating.

If Π is t-graded valid, t-graded consistent, t-live and t-terminating, we say that Π is t-secure.

Definition 8 ((A)synchronous Common Subset (ACS) [BKL20])**.** Let Π be a protocol executed by n parties $P_1, \ldots, P_n$, where each party P_i begins holding input $v_i \in \{0,1\}^*$ and parties output sets of size at most n. The following notions hold whenever there are at most t of the parties are corrupted.

- **Validity**: Π is *t-valid* if the following holds: if every honest party's input is equal to the same value v, then every honest party outputs v.
- **Consistency**: Π is *t-consistent* if all honest parties output the same set S.
- **Liveness**: Π is *t-live* if every honest party produces an output.
- **Set quality**: Π has *t-set quality* if the following holds: if an honest party outputs a set S, then S contains the inputs of at least $t + 1$ honest parties.

Gather is a multi-dealer extension of RBC, where every party is also a dealer. The output of a gather protocol is a *gather-set*.

Definition 9 (Gather-set [AJM+21])**.** A *gather-set* consists of at least $n - t$ pairs (j, x), such that $j \in [n], x \in \mathcal{M}$, and each index j appears at most once, where t is the number of corruptions an adversary can make. For any given gather-set X, define its *index-set* $Indices(X) = \{j | \exists (j, x) \in X\}$ to be the set of the indices that appear in X.

The goal of the gather protocol is to output some common *core gather-set* X^* such that all parties output a super-set of this core. **Note that a gather protocol does not solve consensus and different parties may output different super-sets of the core.**

For *verifiable gather*, the goal is to limit the power of the adversary to generate inconsistent outputs. Intuitively, for any gather-set produced by the adversary, if it passes some verification protocol, it must also be a super-set of the common core.

Definition 10 (Verifiable Gather [AJM+21])**.** A *verifiable gather* protocol consists of a pair of protocols (Gather, Verify) and takes as input an external validity function Validate which all parties have access to. For Gather, each party $i \in [n]$ has an externally valid input x_i. Each party may decide to output a gather-set X_i. After outputting the gather-set, parties must continue to update their local state according to the Gather protocol in order for the verification protocol to continue working. The properties of Gather (assuming a nonfaulty start) are:

- **Binding core**: Once the first nonfaulty party outputs a value from the Gather protocol, there exists a core gather-set X^* such that if a nonfaulty party i outputs the gather set X_i, then $X^* \subseteq X_i$.
- **Internal validity**: If $(j, x) \in X^*$ and j is nonfaulty at the time the first nonfaulty party completed the Gather protocol, then x is the input of party j in Gather.
- **Termination of output**: All nonfaulty parties eventually output a gather-set.

The Verify protocol receives an index-set I and outputs a gather-set X such that $Indices(X) = I$. Verify performs two actions at once: it verifies that the index set includes the indices of the binding core, and recovers the gather-set only from the indices and the internal state of the verifying party. A party i can check any index-set I, which is denoted by executing $\mathsf{Verify}_i(I)$. If the execution of $\mathsf{Verify}_i(I)$ terminates and outputs a value, we say that i has verified the index-set I. The termination properties of Verify (given that all nonfaulty parties start Gather) are:

- **Completeness**: For any two nonfaulty parties i and j, if j outputs X_j from Gather, then $\mathsf{Verify}_i(Indices(X_j))$ eventually terminates with the output X_j.
- **Agreement on verification**: For any two nonfaulty parties i and j, and any index-set I, if $\mathsf{Verify}_i(I)$ terminates with the output X, then $\mathsf{Verify}_j(I)$ eventually terminates with the output X.

The correctness properties of Verify are:

- **Agreement**: All nonfaulty parties agree on values with common indices. For any two nonfaulty i, j, and any index-sets I, J, if $\mathsf{Verify}_i(I)$ terminates with the output X and $\mathsf{Verify}_j(J)$ terminates with the output Y, and $(k, x) \in X, (k, y) \in Y$, then $x = y$.
- **Includes core**: If $\mathsf{Verify}_i(I)$ terminates with the output X, then the gather-set X contains the binding core gather-set X^*.
- **External validity**: If $\mathsf{Verify}_i(I)$ terminates with the output X for some nonfaulty party i, then for each $(j, x) \in X$, the value x is externally valid.

Atomic broadcast guarantees a total order on messages such that honest parties deliver all messages with a common tag in the same order.

Definition 11 (Atomic Broadcast [MXC+16])**.** An atomic broadcast protocol must satisfy the following properties, all of which should hold with high probability (as a function $1 - \mathrm{negl}(\kappa)$ of a security parameter, κ) in an asynchronous network and in spite of an arbitrary adversary:

- **Agreement**: If any correct node outputs a transaction tx, then every correct node outputs tx.
- **Total Order**: If one correct node has output the sequence of transactions $\langle \mathsf{tx}_0, \mathsf{tx}_1, \ldots, \mathsf{tx}_j \rangle$ and another has output $\langle \mathsf{tx}'_0, \mathsf{tx}'_1, \ldots, \mathsf{tx}'_{j'} \rangle$, then $\mathsf{tx}_i = \mathsf{tx}'_i$ for all $i \leq \min(j, j')$.
- **Censorship Resilience**: If a transaction tx is input to $n - t$ correct nodes, then it is eventually output by every correct node.

Secure causal atomic broadcast (SC-ABC) is a useful protocol for building secure applications that use state machine replication in a Byzantine setting. It provides atomic broadcast, which ensures that all recipients receive the same sequence of messages, and also guarantees that the payload messages arrive in an order that maintains "input causality" [CKPS01].

Definition 12 (Byzantine Atomic Broadcast [KKKNS21])**.** Each correct process $p_i \in \Pi$ can call $a\_bcast(m, r)$ and output $a\_deliver_i(m, r, k)$, $p_k \in \Pi$. A Byzantine Atomic Broadcast (BAB) protocol satisfies:

- **Agreement**: If a correct processes $p_i \in \Pi$ outputs $a\_deliver_i(m, r, k)$, then every other correct process $p_j \in \Pi$ eventually outputs $a\_deliver_j(m, r, k)$ with probability 1.
- **Integrity**: For each round $r \in \mathbb{N}$ and process $p_k \in \Pi$, a correct process $p_i \in \Pi$ outputs $a\_deliver_i(r, m, k)$ at most once regardless of m.
- **Validity**: If a correct process $p_k \in \Pi$ calls $a\_bcast_k(m, r)$, then every correct processes $p_i \in \Pi$ eventually outputs $a\_deliver_i(m, r, k)$ with probability 1.
- **Total order**: If a correct process $p_i \in \Pi$ outputs $a\_deliver_i(m, r, k)$ before $a\_deliver_i(m', r', k')$, then no correct process $p_j \in \Pi$ outputs $a\_deliver_j(m', r', k')$ without first outputting $a\_deliver_j(m, r, k)$.

A reliable broadcast protocol satisfies the above agreement, integrity, and validity properties (with messages r_bcast and $r\_deliver$).

The above validity property requires that all messages broadcast by correct processes are eventually ordered (with probability 1), whereas most Byzantine SMR protocols require that in an infinite run, an infinite number of decisions are made, but some proposals by correct processes can be ignored. In addition, the above definition for a BAB protocol satisfies chain quality: for every prefix of ordered messages of size $r(2f + 1), r \in \mathbb{N}$ at least $r(f + 1)$ were broadcast by correct processes.

Protocols for atomic broadcast (ABC) allow parties to maintain agreement on an ever-growing, ordered sequence of blocks, where a block is a set of values called transactions. An ABC protocol does not terminate but instead continues indefinitely. For the ABC protocol below, model the sequence of blocks output by a party P_i via a write-once array $\mathsf{Blocks}_i = \mathsf{Blocks}_i[1], \mathsf{Blocks}_i[2], \ldots$ maintained by P_i, each entry (or slot) of which is initially equal to $\perp$. We say that P_i outputs a block in slot j when P_i writes a block to $\mathsf{Blocks}_i[j]$; if $\mathsf{Blocks}_i[j] \neq \perp$ then we call $\mathsf{Blocks}_i[j]$ the block output by P_i in slot j.

Assume that each party P_i maintains a write-once array $\mathsf{Epochs}_i = \mathsf{Epochs}_i[1], \mathsf{Epochs}_i[2], \ldots$, each entry of which is initialized to 0. We say P_i enters epoch j when it sets $\mathsf{Epochs}_i[j] := 1$, and require: for epoch $j > 1$, P_i enters epoch $j - 1$ before entering epoch j; and P_i enters epoch j before outputting a block in slot j.

An SMR protocol is run in a setting where parties asynchronously receive inputs (i.e., transactions) as the protocol is being executed; each party P_i stores transactions it receives in a local buffer buf_i. We imagine these transactions as being provided to parties by some mechanism external to the protocol (which could involve a gossip protocol run among the parties themselves), and make no assumptions about the arrival times of these transactions at any of the parties.

Definition 13 (Atomic Broadcast (ABC) [BKL20])**.** An atomic broadcast protocol is a protocol Π in which parties $P_1, \ldots, P_n$ are provided with transactions as input and locally maintain arrays Blocks and Epochs. The following notions hold whenever there are at most t of the parties are corrupted.

- **Consistency**: Π is t-consistent if an honest party outputs a block B in slot j then all parties that remain honest output B in slot j.

- **Strong liveness**: Π is t-live if for any transaction tx for which every honest party received tx before entering epoch j, every party that remains honest outputs a block that contains tx in some slot $j' \leq j$.
- **Completeness**: Π is t-complete if for all $j \geq 0$, every party that remains honest outputs some block in slot j.

If Π is t-consistent, t-strongly live and t-complete, we say that Π is t-secure.

ABC with strong liveness implies weak BA (every honest party outputs a value, but may not terminate).

Definition 14 (State Machine Replication (SMR) [MR21])**.** A state machine replication protocol is a protocol in which a replica or client outputs a sequence of transactions $\mathsf{log} = \langle \mathsf{tx}_0, \mathsf{tx}_1, \ldots, \mathsf{tx}_j \rangle$ if and only if there exists a publicly verifiable proof π such that $\mathsf{Verify}(\mathsf{log}, \pi) = 1$, where Verify is a Boolean function known to all parties. An SMR protocol must additionally satisfy the following:

- **Safety**: If $\langle \mathsf{tx}_0, \mathsf{tx}_1, \ldots, \mathsf{tx}_j \rangle$ and $\langle \mathsf{tx}'_0, \mathsf{tx}'_1, \ldots, \mathsf{tx}'_{j'} \rangle$ are output by two honest replicas or clients, then $\mathsf{tx}_i = \mathsf{tx}'_i$ for all $i \leq \min(j, j')$.
- **Liveness**: If a transaction tx is input to at least one honest replica, then every honest replica eventually outputs a log containing tx.

Definition 15 (State Machine Replication with batching and commit [MR21])**.** A state machine replication protocol is a protocol in which a replica or client outputs a sequence of transactions $\mathsf{log} = \langle \mathsf{tx}_0, \mathsf{tx}_1, \ldots, \mathsf{tx}_j \rangle$ if and only if there exists a publicly verifiable proof π such that $\mathsf{Verify}(\mathsf{log}, \pi) = 1$, where Verify is a Boolean function known to all parties.

Transactions batched within a *block* ($\mathsf{B} = \langle \mathsf{tx}_0, \mathsf{tx}_1, \ldots, \mathsf{tx}_j \rangle$) are totally ordered, so a log of blogs can be flattened in a log of transactions (i.e., the log of blocks B and B' can be interpreted as a log of transactions $\langle \mathsf{tx}_0, \mathsf{tx}_1, \ldots, \mathsf{tx}_j, \mathsf{tx}'_0, \mathsf{tx}'_1, \ldots, \mathsf{tx}'_{j'} \rangle$).

The publicly verifiable proof for a block serves as the public verifiable proof for each transaction in that block.

A replica *commits* to a new block by broadcasting a signature on the block. In the paradigm of chained SMR, the last block of a log uniquely identifies the entire log (hence, signing a block is equivalent to signing the entire log up to the block).

An SMR protocol must additionally satisfy the following:

- **Safety**: If two honest replicas commit two logs $\langle \mathsf{B}_0, \mathsf{B}_1, \ldots, \mathsf{B}_j \rangle$ and $\langle \mathsf{B}'_0, \mathsf{B}'_1, \ldots, \mathsf{B}'_{j'} \rangle$, then $\mathsf{B}_i = \mathsf{B}'_i$ for all $i \leq \min(j, j')$.
- **Liveness**: Every transaction is eventually committed by all honest replicas.

Definition 16 (Blockchain protocol [Shi20, Ch. 6])**.** In a *blockchain protocol*, a set of distributed nodes aim to agree on an ever-growing, linearly-ordered log of transactions. The nodes each maintain a growing linearly ordered log of transactions. Denote node i's log in round r by LOG_i^r (which is also called a *finalized log*, i.e., every transaction or event contained in LOG_i^r cannot be undone later). A blockchain protocol must satisfy the following two properties:

- **Consistency**: All nodes have the same view of the linearly ordered log (despite the potential delay), i.e., the honest nodes' logs are prefixes of each other. More formally: for any honest nodes i and j, and for any round numbers t and r, it must be that $\mathsf{LOG}_i^t \preceq \mathsf{LOG}_j^r$ or $\mathsf{LOG}_i^t \succeq \mathsf{LOG}_j^r$.
- **Liveness**: If an honest node receives some transaction tx as input in some round r, then by the end of round $r + T_{conf}$, all honest nodes' local logs must include tx.

Assuming the existence of a PKI and digital signatures, a blockchain protocol can be constructed by sequential composition of Byzantine broadcast.

Definition 17 (Bitcoin Backbone [GKL20])**.** Consider a chain $\mathcal{C}$ of length m and any nonnegative integer k. We denote by $\mathcal{C}^{\lceil k}$ the chain resulting from the "pruning" of the k rightmost blocks. Note that for $k \geq \mathrm{len}(\mathcal{C})$, $\mathcal{C}^{\lceil k}$ is the empty string. If $\mathcal{C}_1$ is a prefix of $\mathcal{C}_2$, we write $C_1 \preceq C_2$.

- k-**Common Prefix Property**: For any pair of honest parties P_1, P_2 adopting the chains $\mathcal{C}_1, \mathcal{C}_2$ at rounds $r_1 \leq r_2$, it holds that $\mathcal{C}_1^{\lceil k} \preceq \mathcal{C}_2$.
- (μ, ℓ)-**Chain Quality Property**: For any honest party P with chain $\mathcal{C}$, the ratio of adversarial blocks among any ℓ consecutive blocks of $\mathcal{C}$ is at most μ.
- (τ, s)-**Chain Growth Property**: For any honest party P with chain $\mathcal{C}$, it holds that after s consecutive rounds it adopts a chain that is at least $\tau \cdot s$ longer than the initial chain $\mathcal{C}$.

Definition 18 (Robust Public Transaction Ledger [GKL20])**.** A protocol Π implements a *robust public transaction ledger* in the q-bounded synchronous setting (where q is the number of brute force queries a party can make in a round) if it organizes the ledger as a hashchain of transactions and it satisfies the following two properties:

- k-**Persistence** If in some round r, an honest player reports a ledger that contains a transaction tx in a block more than k blocks away from the end of the ledger (such transactions are called "stable"), then tx will be reported by any honest player in the same position of the ledger for any round $r' \geq r$.
- (u, k)-**Liveness** Provided a non-conflicting transaction tx is given as input to all honest players continuously for u consecutive rounds, then all honest parties will report tx more than k block from the end of the ledger, i.e., all report it as stable.

2 Possibility and impossibility results

2.1 Synchronous setting

2.1.1 Corruptions and (non)determinism

- With no prior setup, BB is possible when $t < n/3$ parties are corrupted.
- With no prior setup, BB is impossible when $t \geq n/3$ parties are corrupted.
- If the parties have a pre-existing PKI, then BB is possible when $t < n$ parties are corrupted.
- The optimal resilience for BA and BFT SMR in the synchronous authenticated setting is $t < n/2$.
- For $t < n/2$, BA implies BB and BB implies BA. Blockchain implies BA [GKL15] but for $1/3$ adversarial power. Blockchain implies BB using proof of work [GKL15] for honest majority.
- Under PKI setup, SMR implies RBC (under the $t < n/3$ faults in both synchronous and asynchronous cases) [MR21].
- Strong-termination SMR implies weak BA [BKL20].
- Byzantine consensus can be solved with $t < n$ when using signatures. However, it is necessary to have $t < n/3$ to solve this problem with symmetric authentication [PSL80].

2.1.2 Communication complexity

- Lamport, Shostak and Pease: BB for $t < n/3$ with exponential communication complexity (setup free).
- King et al: BB for $t < n/3$ with $\tilde{O}(n^{3/2})$ communication complexity, Boyle et al $\tilde{O}(1)$ balanced communication (setup free).
- Micali, Vaikuntanathan and Micali: BB for $n/3 < t < n/2$ with subquadratic communication complexity (trusted setup).
- Dolev and Strong: BB for $t < n$ with $O(n^3\kappa)$ communication complexity (bulletin PKI).
- Abraham et al: BB for $t < n/2$ adaptive corruptions with $\tilde{O}(n\kappa$ communication complexity (trusted setup).

- Chan et al: BB for adaptive $t < (1-\epsilon)n$ corruptions with $O(n^2\kappa^2$ communication complexity (trusted PKI).
- Momose and Ren: BB for $t < n/2$ adaptive corruptions with $\tilde{O}(n^2\kappa)$ communication complexity (bulletin PKI).
- Tsimos, Loss and Papamanthou: Random BB for $t < (1-\epsilon)n$ static corruptions with $O(n^2\kappa^2)$ communication complexity and $O(t+1+\log n)$ round complexity (bulletin PKI) and Parallel BB for $t < (1-\epsilon)n$ adaptive corruptions with $O(n^2\kappa^4)$ communication complexity (trusted PKI).

2.2 Asynchronous setting

- Every asynchronous protocol solving BA has executions that do not terminate [FLP85]. Hence, deterministic asynchronous atomic broadcast is impossible [MXC+16]. SMR in asynchronous networks is also subject to this limitation.
- With prior setup, randomized algorithms for solving BB, BA, SMR exist for $t < n/3$.
- No partially synchronous protocol can realize BA for $t \geq n/3$ [DLS88] even with setup (and with probability greater than $2/3$).
- For $t < n/3$, BA implies BB and BB implies BA? But RBC does not imply BA.
- Strong-termination SMR implies weak BA [BKL20].
- (Binary) Byzantine agreement and atomic broadcast are equivalent in the asynchronous setting, assuming a secure signature scheme and provided $t < n/3$ (first stated in [CT96, p. 248] for crash faults and proved in [Sec. 5.3][CKPS01]). Note that using an appropriately defined notion of authenticated atomic broadcast, this could also be implemented without the additional digital signatures in the reduction.
- BA with weak validity only does not solve Atomic Broadcast or State Machine Replication (SMR) [AMS19].
- Simple reduction between BA and Atomic Broadcast requires a strong validity property for asynchronous networks [Tang,Lu,Lu,Wang PODC 2020]. It is intractable to get strong validity asynchronous BA.
- The existence of a partially synchronous blockchain protocol implies the existence of a partially synchronous BA protocol, for $t < n/3$. In the reverse direction, given a partially synchronous BA protocol resilient against t corrupt nodes, we can construct a partially synchronous blockchain protocol resilient against t corrupt nodes [Shi20] (for arbitrary t).
- Any protocol solving verifiable secret sharing in the asynchronous model with optimal resilience must have a positive probability of deadlock [BOKR94].

3 Ideal functionalities

Synchronous broadcast [HZ10].

Functionality $\mathcal{F}_{BC}$

1. P_s sends its input x_s to functionality $\mathcal{F}_{BC}$.
2. $\mathcal{F}_{BC}$ sends x_s to all parties $P \in \mathcal{P}$.

Synchronous broadcast without secrecy [GKKZ11].

Functionality $\mathcal{F}_{BC}$

1. Upon receiving $(\mathsf{Bcast}, \mathsf{sid}, m)$ from P_i, send $(\mathsf{Bcast}, \mathsf{sid}, P_i, m)$ to all parties $P \in \mathcal{P}$ and to the adversary $\mathcal{S}$.

Synchronous unfair broadcast [HZ10].

Functionality $\mathcal{F}_{UBC}$

1. P_s sends its input x_s to functionality $\mathcal{F}_{UBC}$.
2. $\mathcal{F}_{UBC}$ sends x_s to the adversary.
3. If P_s is corrupted then the adversary sends a value to $\mathcal{F}_{UBC}$; $\mathcal{F}_{UBC}$ denotes the received value by x'_s (if P_s is not corrupted then $\mathcal{F}_{UBC}$ sets $x'_s := x_s$).
4. $\mathcal{F}_{UBC}$ sends x'_s to all parties $P \in \mathcal{P}$.

Synchronous relaxed broadcast [GKKZ11] (reformulation of $\mathcal{F}_{UBC}$).

Functionality $\mathcal{F}_{RelBC}$

1. Upon receiving $(\mathsf{Bcast}, \mathsf{sid}, m)$ from P_i, send $(\mathsf{Bcast}, \mathsf{sid}, P_i, m)$ to the adversary $\mathcal{S}$.
2. Upon receiving m' from the adversary $\mathcal{S}$, do:
   - If P_i is corrupted, send $(\mathsf{Bcast}, \mathsf{sid}, P_i, m')$ to all parties $P \in \mathcal{P}$.
   - If P_i is not corrupted, send $(\mathsf{Bcast}, \mathsf{sid}, P_i, m)$ to all parties $P \in \mathcal{P}$.

Synchronous UC-Byzantine Agreement as a Canonical Synchronous Functionality (functionality without probabilistic termination) [CCGZ19].

Functionality $\mathcal{F}_{BA}^{V}$

Each party $P_i \in \mathcal{P}$ has input $x_i \in V$.

1. Initially, set the input values $x_1, \ldots, x_n$, the output values $y_1, \ldots, y_n$ and the adversary's value to $\perp$.
2. In round $\rho = 1$:
   - Upon receiving $(\mathsf{adv\text{-}input}, \mathsf{sid}, v)$ from the adversary, set $a \leftarrow v$.
   - Upon receiving a message $(\mathsf{input}, \mathsf{sid}, v)$ from some party $P_i \in \mathcal{P}$, set $x_i \leftarrow v$ and send $(\mathsf{leakage}, \mathsf{sid}, P_i, x_i)$ to the adversary.
3. In round $\rho = 2$:
   - Upon receiving $(\mathsf{adv\text{-}input}, \mathsf{sid}, v)$ from the adversary, if $y_1 = \ldots = y_n = \perp$, set $a \leftarrow v$. Otherwise, discard the message.
   - Upon receiving a message $(\mathsf{fetch\text{-}output}, \mathsf{sid}, v)$ from some party $P_i \in \mathcal{P}$, if $y_1 = \ldots = y_n = \perp$ compute $(y_1, \ldots, y_n) = (y, \ldots, y)$ such that $y = x$ if there exists a $x = x_i$ for $n - t$ input values x_i, otherwise $y = a$. Next, send $(\mathsf{output}, \mathsf{sid}, y)$ to P_i and $(\mathsf{fetch\text{-}output}, \mathsf{sid}, P_i)$ to the adversary.

We need wrappers around $\mathcal{F}_{BA}^{V}$ if we want to simulate probabilistic termination.
ABC from [BKL20] (see Definition 13).

Functionality $\mathcal{F}_{ABC}$

1. For each party P_i, instantiate $\mathsf{Epoch}_i[j] = 0$, for $j \geq 1$. Initialize $j = 1$.
2. While true do:
   (a) Set $\mathsf{Epoch}_i[j] = 1$.
   (b) Upon receiving $n - t$ inputs $(\mathsf{input}, \mathsf{buf}_i, j, P_i)$ for distinct i, set $B_j = \bigcup_{P_i \in \mathcal{P}^{received}} \mathsf{buf}_i$.
   (c) For each i, set $\mathsf{Blocks}_i[j] = B_j$ and output $(\mathsf{output}, \mathsf{Blocks}_i[j], j, P_i)$.
   (d) $j = j + 1$.

In the first epoch, each party sends its first transaction and the adversary decides if it corrupts the party and change the input. For epochs $j > 1$, after receiving $(\mathsf{output}, \mathsf{Blocks}_i[1], 1, P_i), \ldots, (\mathsf{output}, \mathsf{Blocks}_i[j-1], j-1, P_i)$, party P_i sends $(\mathsf{input}, \mathsf{buf}_i, j, P_i)$ to $\mathcal{F}_{ABC}$, and the adversary decides if it corrupts the party and change the input.

References

[AJM+21] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. Reaching consensus for asynchronous distributed key generation. *arXiv preprint arXiv:2102.09041*, 2021.

[AMS19] Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. Asymptotically optimal validated asynchronous byzantine agreement. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 337–346, 2019.

[BKL19] Erica Blum, Jonathan Katz, and Julian Loss. Synchronous consensus with optimal asynchronous fallback guarantees. In *Theory of Cryptography Conference*, pages 131–150. Springer, 2019.

[BKL20] Erica Blum, Jonathan Katz, and Julian Loss. Network-agnostic state machine replication. *arXiv preprint arXiv:2002.03437*, 2020.

[BOKR94] Michael Ben-Or, Boaz Kelmer, and Tal Rabin. Asynchronous secure computations with optimal resilience. In *Proceedings of the thirteenth annual ACM symposium on Principles of distributed computing*, pages 183–192, 1994.

[CCGZ19] Ran Cohen, Sandro Coretti, Juan Garay, and Vassilis Zikas. Probabilistic termination and composability of cryptographic protocols. *Journal of Cryptology*, 32(3):690–741, 2019.

[CKPS01] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In *Annual International Cryptology Conference*, pages 524–541. Springer, 2001.

[CL99] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[CR93] Ran Canetti and Tal Rabin. Fast asynchronous byzantine agreement with optimal resilience. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 42–51, 1993.

[CT96] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM (JACM)*, 43(2):225–267, 1996.

[DLS88] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)*, 35(2):288–323, 1988.

[FLP85] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.

[GKKZ11] 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.

[GKL15] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 281–310. Springer, 2015.

[GKL20] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. Cryptology ePrint Archive, Report 2014/765, 2020. https://ia.cr/2014/765.

[HZ10] 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.

[KK09] Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for byzantine agreement. *Journal of Computer and System Sciences*, 75(2):91–112, 2009.

[KKKNS21] Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. All you need is dag. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, PODC'21, pages 165–175. Association for Computing Machinery (ACM), 2021.

[LM18] Julian Loss and Tal Moran. Combining asynchronous and synchronous byzantine agreement: The best of both worlds. *IACR Cryptol. ePrint Arch.*, 2018:235, 2018.

[MR21] Atsuki Momose and Ling Ren. Multi-threshold byzantine fault tolerance. Cryptology ePrint Archive, Report 2021/671, 2021. https://eprint.iacr.org/2021/671.

[MXC+16] 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*, pages 31–42, 2016.

[Shi20] Elaine Shi. Foundations of distributed consensus and blockchains, 2020.