Byzantine Fault-Tolerant Causal Order Satisfying Strong Safety

Anshuman Misra and Ajay D. Kshemkalyani(✉)

University of Illinois at Chicago, Chicago, IL 60607, USA
{amisra7,ajay}@uic.edu

Abstract. Causal ordering is an important building block for distributed software systems. It was recently proved that it is impossible to provide causal ordering – liveness and strong safety – using a deterministic non-cryptographic algorithm in the presence of even a single Byzantine process in an asynchronous system for unicast, multicast, and broadcast modes of communication. Strong safety is critical for real-time distributed collaborative software such as multiplayer gaming and social media networks. In this paper, we solve the causal ordering problem under the strong safety condition in the presence of Byzantine processes by relaxing the problem specification in two ways. First, we propose a deterministic algorithm for causal ordering of unicasts in a synchronous system that also uses threshold cryptography. Second, we propose a (probabilistic) algorithm based on randomization for causal ordering of multicasts in an asynchronous system that also uses threshold cryptography. These algorithms complement the previous impossibility result for the asynchronous system.

Keywords: Causal Order · Message Passing · Byzantine Fault-Tolerance · Distributed Systems · Multicast

1 Introduction

Many distributed applications rely on causal ordering of messages for correct semantics [2,14,15]. Algorithms for providing causal ordering have been proposed over nearly the last four decades. Causal ordering requires that liveness (each message sent by a correct process to another correct process is eventually delivered) and strong safety (if the send event for message m_1 happens before the send event for message m_2 and both messages are sent to the same correct process(es), no correct process delivers m_2 before m_1) are satisfied.

It was recently proved that it is impossible to provide causal ordering – liveness and strong safety – (using a deterministic algorithm) in the presence of even a single Byzantine process in an asynchronous system for unicast, multicast, and broadcast modes of communication in a system model that does not allow cryptography [21,22]. In light of this result, algorithms for Byzantine-tolerant causal ordering under the synchronous system model that satisfy liveness and

S. Dolev and B. Schieber (Eds.): SSS 2023, LNCS 14310, pp. 111–125, 2023.
https://doi.org/10.1007/978-3-031-44274-2_10

a weaker notion of safety, namely *weak safety*, wherein there is path from the send event of m_1 to the send event of m_2 passing through only correct processes, were proposed [19,21]. These algorithms were for unicast, multicast, as well as broadcast modes of communication. For the broadcast mode of communication, a Byzantine-tolerant causal ordering algorithm for asynchronous systems was proposed in [1] – this satisfies liveness and weak safety but no strong safety as shown in [21]. Previously, a probabilistic algorithm based on atomic (total order) broadcast and cryptography for secure causal atomic broadcast (liveness and strong safety) in an asynchronous system was proposed [5]. This logic used acknowledgements and effectively processed the atomic broadcasts serially. More recently for the client-server configuration, two protocols for crash failures and a third for Byzantine failure of clients based on cryptography were proposed for secure causal atomic broadcast [9]. The third made assumptions on latency of messages, and hence works only in a synchronous system.

Main Contributions: The impossibility result given in [22] showed a reduction from consensus to causal ordering, and the FLP impossibility result for consensus [11] implied the impossibility of causal ordering using a deterministic algorithm in an asynchronous system. Solving consensus is equivalent to or mutually reducible to solving the atomic broadcast problem [24], and both are impossible using deterministic algorithms in an asynchronous system. In this paper, we overcome the impossibility result of [21,22] mentioned above. We solve the causal ordering problem in the presence of Byzantine processes by relaxing the system assumptions in two ways.

1. First, we weaken the asynchrony assumption and propose an algorithm to solve the causal ordering problem under the strong safety condition for unicasts in a synchronous system that also uses threshold cryptography.
2. Second, we propose an algorithm based on atomic broadcast in an asynchronous system having Byzantine processes; the algorithm also uses threshold cryptography. Solving consensus is equivalent to or mutually reducible to solving the atomic broadcast problem [6,18], and both are impossible using deterministic algorithms in an asynchronous system. However, atomic broadcasts, i.e., total order broadcasts, can be solved (in the presence of Byzantine processes) only using probabilistic algorithms in an asynchronous system [5,7,12,16]. Our second algorithm for causal ordering uses a source-order preserving total order broadcast primitive as a lower layer interface. It uses threshold cryptography similar to the way it is used in [5] for secure causal atomic broadcast but does not use acknowledgements and is not constrained to process the atomic broadcasts serially, thus there is no concurrency inhibition. Our algorithm is presented for the multicast mode of communication and we show how it can be modified to unicast and broadcast modes which are special cases of multicast mode.

Our algorithms complement the previous impossibility result for the asynchronous system. The main contribution of this paper is to develop efficient causal ordering algorithms that provide *strong safety* in the presence of Byzantine processes. These algorithms bypass the impossibility result proved in [21,22], which states that it is impossible to provide Byzantine-tolerant strong safety in the absence of cryptographic protocols. This is a critical result from the perspective of real-world distributed applications because weak safety cannot guarantee correct functioning of applications. For example, in a multiplayer online gaming scenario utilizing a weak safety protocol for causal ordering, Byzantine players can order their events ahead of correct players' events despite having causal dependencies on the correct players' events leading to unfair advantages in gameplay. However, by using a strong safety causal ordering algorithm, the gaming application can ensure fair gameplay. Similar situations can arise in social media networks (message ordering presented to users in a single message thread), collaborative group editing of documents (updates to documents need to ensure causality across updates regardless of whether the update comes from a correct/Byzantine user) among other distributed applications.

Outline: Section 2 gives the system model. Section 3 reviews some basic cryptography used in our algorithms. Section 4 reviews the specifications of Byzantine-tolerant reliable multicast/broadcast and Byzantine-tolerant atomic broadcast. Section 5 gives the algorithm for Byzantine-tolerant causal order of unicasts in a synchronous system. Section 6 gives the algorithm for Byzantine-tolerant causal order of multicasts in an asynchronous system. Section 7 concludes.

2 System Model

This paper deals with a distributed system having Byzantine processes which are processes that can misbehave [17,23]. A correct process behaves exactly as specified by the algorithm whereas a Byzantine process may exhibit arbitrary behaviour including crashing at any point during the execution. A Byzantine process cannot impersonate another process or spawn new processes.

The distributed system is modelled as an undirected graph $G = (P, H)$. Here P is the set of processes communicating asynchronously in the distributed system. Let n be $|P|$. H is the set of FIFO logical communication links over which processes communicate by message passing. G is a complete graph.

The system is first assumed to be synchronous, i.e., there is a known fixed upper bound δ on the message latency, and a known fixed upper bound ψ on the relative speeds of processors [10]. We provide a deterministic causal ordering unicast algorithm for this system model. Next, we assume an asynchronous system, i.e., there is no upper bound δ on the message latency, nor any upper bound ψ on the relative speeds of processors [10]. We provide a non-deterministic causal ordering multicast algorithm for this system model.

Definition 1. *The happens before relation $\rightarrow$ on messages consists of the following rules:*

1. *The set of messages delivered from any $p_i \in P$ by a process is totally ordered by $\rightarrow$.*
2. *If p_i sent or delivered message m before sending message m', then $m \rightarrow m'$.*

3. If $m \rightarrow m'$ and $m' \rightarrow m''$, then $m \rightarrow m''$.

Let R denote the set of messages in the execution.

Definition 2. *The causal past of message m is denoted as $CP(m)$ and defined as the set of messages in R that causally precede message m under $\rightarrow$.*

The correctness of Byzantine causal order unicast/multicast/broadcast is specified on $(R, \rightarrow)$ for strong safety.

Definition 3. *A causal ordering algorithm for unicast/multicast/broadcast messages must ensure the following:*

*1. **Strong Safety:** $\forall m' \in CP(m)$ such that m' and m are sent to the same (correct) process(es), no correct process delivers m before m'.*
*2. **Liveness:** Each message sent by a correct process to another correct process will be eventually delivered.*

3 Some Cryptographic Basics

We utilize non-interactive threshold cryptography as a means to guarantee strong safety [25]. Threshold cryptography consists of an initialization function to generate keys, message encryption, sharing decrypted shares of the message and finally combining the decrypted shares to obtain the original message from ciphertext. The following functions are used in a threshold cryptographic scheme:

Definition 4. *The dealer executes the generate() function to obtain the public key PK, Verification key VK and the private keys SK_1, SK_2, ... , SK_n.*

The dealer shares private key SK_i with each process p_i while PK and VK are publicly available.

Definition 5. *When process p_i wants to send a message m to p_j, it executes $E(PK, m, L)$ to obtain C_m. Here C_m is the ciphertext corresponding to m, E is the encryption algorithm and L is a label to identify m. p_i then broadcasts C_m to the system of processes.*

Definition 6. *When process p_l receives ciphertext C_m, it executes $D(SK_l, C_m)$ to obtain σ_l^m where D is the decryption share generation algorithm and σ_l^m is p_l's decryption share for message m.*

When process p_j receives a cipher message C_m intended for it, it has to wait for k decryption shares to arrive from the system to obtain m. The value of k depends on the security properties of the system. It derives the message from the ciphertext as follows:

Definition 7. *When process p_j wants to generate the original message m from ciphertext C_m, it executes $C(VK, C_m, S)$ where S is a set of k decryption shares for m and C is the combining algorithm for the k decryption shares that gives m.*

The following function V is used to verify the authenticity of a decryption share:

Definition 8. *When a decryption share σ is received for message m, the Share Verification Algorithm is used to ascertain whether σ is authentic:*
$V(VK, C_m, \sigma) = 1$ *if σ is authentic,* $V(VK, C_m, \sigma) = 0$ *if σ is not authentic.*

4 Reliable Broadcast and Atomic (Total Order) Broadcast Properties

The multicast algorithm for asynchronous systems that we propose assumes access to a BA_broadcast primitive that provides Byzantine-tolerant total order and delivers a broadcast message via BA_deliver.

Definition 9. *Byzantine-tolerant atomic (total order) broadcast provides the following guarantees [7,8,12,13,16,24]:*

1. *(BAB-Validity:) If a correct process* BA_deliver*s a message m from sender-(m), then sender(m) must have* BA_broadcast *m.*
2. *(BAB-Termination-1:) If a correct process* BA_broadcast*s a message m, then it eventually* BA_deliver*s m.*
3. *(BAB-Agreement or BAB-Termination-2:) If a correct process* BA_deliver*s a message m from a possibly faulty process, then all correct processes eventually* BA_deliver *m.*
4. *(BAB-Integrity:) For any message m, every correct process* BA_deliver*s m at most once.*
5. *(BAB-Total-Order:) If correct processes p_i and p_j both* BA_deliver *messages m and m', then p_i* BA_deliver*s m before m' if and only if p_j* BA_deliver*s m before m'.*

This total order primitive also provides source-FIFO order [13], i.e., if a process BA_broadcasts m before m', then m is BA_delivered before m' at all correct processes. As it is impossible to provide Byzantine-tolerant total order using a deterministic algorithm in an asynchronous system due to its equivalence to consensus [5,24], we use a probabilistic algorithm such as in [5,7,12,16].

Byzantine Reliable Broadcast (BRB) [3,4] is invoked via BR_broadcast and delivered via BR_deliver. It is defined similar to Definition 9 minus BAB-Total-Order.

We propose a causal order multicast algorithm for asynchronous systems. In a multicast, a message is sent to a subset of processes forming a process group. Different multicast send events can send to different process groups. Byzantine-tolerant causal multicast is invoked as BC_multicast(m, G), where G is the multicast group, and delivers a message through BC_deliver(m). Based on the reliability properties proposed in the literature for Byzantine Reliable Broadcast [3,4] and Byzantine Causal Broadcast [1], we define Byzantine Causal Multicast as follows.

Definition 10. *Byzantine Causal Multicast satisfies the following properties:*

1. *(BCM-Validity:) If a correct process p_i* BC_deliver*s message m from sender(m) to group G, then sender(m) must have* BC_multicast *m to G and $p_i \in G$.*
2. *(BCM-Termination-1:) If a correct process* BC_multicast*s a message m to G, then some correct process in G eventually* BC_deliver*s m.*
3. *(BCM-Agreement or BCM-Termination-2:) If a correct process in G* BC_deliver*s a message m from a possibly faulty process, then all correct processes in G will eventually deliver m.*
4. *(BCM-Integrity:) For any message m, every correct process in G* BC_deliver*s m at most once.*
5. *(BCM-Causal-Order:) If $m \to m'$, m is sent to G, m' is sent to G', then no correct process in $G \cap G'$* BC_deliver*s m' before m.*

BCM-Causal-Order is the Strong Safety property of Definition 3 whereas BCM-Termination-1 and BCM-Agreement imply the liveness property of Definition 3.

Definition 11. *A Byzantine-tolerant causal multicast algorithm must satisfy BCM-Validity, BCM-Termination-1, BCM-Agreement, BCM-Integrity, and BCM-Causal-Order.*

5 Causal Order Unicast in a Synchronous System

In Algorithm 1 we present a causal ordering algorithm guaranteeing strong safety and liveness in the presence of t Byzantine processes for synchronous systems. Algorithm 1 is inherently asynchronous, because it does not assume the expensive and binding notion of rounds. Algorithm 1 requires that key generation and distribution has been accomplished by a trusted dealer prior to start of execution. Therefore, all processes have access to global PK (public key), VK (verification key) and have a local SK_i (secret key). Algorithm 1 assumes that the network provides an upper bound δ on the message transmission time. Algorithm 1 has to prevent Byzantine processes from implementing the following actions:

1. Reading the contents of an incoming message prior to delivering it and sending an outgoing message based on the contents of the undelivered message with the intention of causing a strong safety violation.
2. Sending a message to a correct process with the intention of preventing further messages getting delivered at that process, causing a liveness attack.

When p_i wants to unicast a message m to p_j, it encrypts m with PK and broadcasts the ciphertext C_m along with p_j's id (j) and a globally unique message id id_m to the system. p_j requires $(t+1)$ unique decryption shares from processes in the system to obtain m from C_m. Upon receiving a ciphertext C_m, all processes compute their respective decryption shares σ_x^m. Upon receiving C_m, p_j inserts C_m into its FIFO delivery queue and broadcasts a request for decryption shares.

If the required number of decryption shares do not arrive within $(3\delta + 1)$ time units, p_j will delete C_m from its delivery queue preventing liveness attacks by Byzantine processes. This will be formally proved in Theorem 1.

When a process p_k receives p_j's request for its decryption share for m, it first checks to make sure that p_j is indeed the recipient of m. If that is the case, p_k waits for $(\delta + 1)$ time units and sends σ_k^m to p_j. Once p_j receives the required number of decryption shares, it decrypts C_m and replaces it with m in its delivery queue. When C_m is both decrypted and m is at the head of the queue, it gets delivered when the application is ready to process the next message. The intuitive reasoning for preservation of strong safety by Algorithm 1 is as follows: Since correct processes wait for $(\delta+1)$ time units before sending their decryption shares, a Byzantine process p_k can read a message m at least $(\delta + 1)$ time units after receiving C_m. Hence, any message m' such that $m \rightarrow m'$ that p_k sends to process p_l will arrive at p_l at least 1 time unit after any m'' sent to p_l, where $m'' \rightarrow m \rightarrow m'$. Hence m' will be after m'' in the delivery queue at p_l. This is formally proved in Theorem 2. Algorithm 1 can tolerate upto t Byzantine failures, where the total number of processes, $n > 2t$.

Each message m has a globally unique identifier id_m assigned by the sender. In the Algorithm 1 pseudo-code, technically C_m, σ_i^m, S should be C_{id_m}, $\sigma_i^{id_m}$, S_{id_m} respectively; however to simplify the presentation, we use the first version while keeping in mind that the data structure is to be associated with a particular id_m.

Theorem 1. *All messages sent by a correct process to another correct process via unicast following Algorithm 1 will eventually be delivered even in the presence of Byzantine processes.*

Proof. Consider message m sent by p_i to p_j. p_i executes $broadcast(C_m, j, id_m)$ at line 3, ensuring that all processes receive C_m and compute their respective decryption shares at line 5. Once p_j receives C_m, it pushes C_m into a FIFO queue, starts a timer of $(3\delta + 1)$ time units at lines 6–8. p_j then broadcasts a request for decryption shares to all processes at line 9. The maximum latency for an individual response to this request is the sum of (i) the maximum of the maximum time it takes for the request sent at line 9 to arrive (δ) at a receiver and the maximum time it takes for the broadcast of line 3 to reach the receiver (δ), (ii) the waiting time at the receiver of this request $(\delta + 1)$, and (iii) the maximum latency of the response to p_j (δ). Therefore, p_j will receive decryption share σ_x^m from each correct process p_x within $\max(\delta, \delta) + (\delta + 1) + \delta = (3\delta + 1)$ time units. Since there are at least $(t + 1)$ correct processes, p_j is guaranteed to receive the required $(t + 1)$ decryption shares in line 16 before message m times out (lines 20–22). Therefore, C_m is guaranteed to be decrypted and m is guaranteed to be present in Q (lines 16–19).

A ciphertext $C_{m'}$ present ahead of m in Q at p_j is one of the following:

1. $C_{m'}$ was sent by a Byzantine process p_l. In this case, the required number of decryption shares for $C_{m'}$ in lines 16–20 may not arrive within $(3\delta + 1)$ time

Algorithm 1: Secure Causal Unicast in a Synchronous System

```
Data: Each process has access to $PK$ (global public key) and $VK$ (global
      verification key) as well as a local secret key $SK_i$. Each process
      maintains a FIFO queue $Q$ for incoming application messages.
1  when $p_i$ needs to send application message $m$ to $p_j$:
2  $C_m = E(PK, m, id_m)$
3  $broadcast(C_m, j, id_m)$
4  when $\langle C_m, recipient, id_m \rangle$ arrives at $p_i$:
5  $\sigma_i^m = D(SK_i, C_m)$
6  if $recipient = i$ then
7      $Q.push(C_m)$
8      start $timer$ set to $3\delta + 1$ for message $m$
9      $broadcast(request, id_m)$ to $\forall p_x$
10 when $p_i$ receives $\langle request, id_m \rangle$ from $p_j$
11 if $C_m$ has not arrived at $p_i$ then
12     wait for min($\delta$ time units, arrival of $C_m$) in a non-blocking manner
13 if $C_m$ has arrived $\wedge$ $p_j$ is the recipient of message $m$ then
14     wait for $(\delta + 1)$ time units in a non-blocking manner
15     send($\sigma_i^m$) to $p_j$
16 when $p_i$ receives $(t + 1)$ valid $\langle \sigma_x^m \rangle$ messages:
17 Store $(t + 1)$ decryption shares in set $S$
18 $m = C(VK, C_m, S)$
19 replace $C_m$ in $Q$ with $m$
20 when any $C_m$ times out in $Q$:
21 if less than $(t+1)$ valid decryption shares corresponding to $m$ have arrived then
22     $Q.delete(C_m)$
23 when the application is ready to process a message at $p_i$:
24 if $Q.head()$ is decrypted then
25     $m = Q.pop()$
26     deliver $m$
```

units since starting the timer for $C_{m'}$. In this case $C_{m'}$ will be deleted from the queue in lines 20–22, thus ensuring progress.

2. $C_{m'}$ was sent by a correct process p_k. Therefore, within $(3\delta + 1)$ time units since its insertion in Q at p_j, $C_{m'}$ will be decrypted and m' will be present in Q ready to be delivered as p_k and correct processes will follow the protocol.

Combining points 1 and 2, m is guaranteed to reach the head of Q and eventually be delivered in lines 23–26. □

Corollary 1. *Algorithm 1 guarantees liveness.*

Theorem 2. *If $m_1 \rightarrow m_2$ and both messages are sent to the same correct destination process, then Algorithm 1 guarantees that m_2 is not delivered before m_1.*

Proof. Consider messages m_1 and m_2 sent to a correct process p_k where $m_1 \rightarrow m_2$. In order for p_k to ensure causal delivery of m_1 with respect to m_2 (lines 23–26), C_{m_1} must be enqueued in Q before C_{m_2} in lines 4–9. One of the following scenarios must hold:

1. The same process p_i sent both m_1 and m_2. Due to FIFO channels, C_{m_1} will arrive before C_{m_2} at p_k and as a result, get enqueued in Q before C_{m_2}.
2. p_i sent m_1 and p_j sent m_2. As $m_1 \rightarrow m_2$, there must be at least one message hop along the message chain from the sending of m_1 to the sending of m_2. Let the last message along this message chain, which was delivered to p_j, be m^*. A lower bound on the duration between the sending of m_1 and the sending of m_2 is $(\delta + 1)$. This is because C_{m^*} must have resided in the Q at p_j at least for $(\delta + 1)$ time units, the duration that p_j's request is delayed by the correct processes before they send p_j their decryption shares for C_{m^*}, before C_{m^*} is decrypted and delivered to p_j.
   As m_2 is sent at least $(\delta + 1)$ time units after m_1 is sent to the common destination p_k, even if C_{m_1} takes the full δ time units to reach p_k and C_{m_2} takes 0 time units to reach p_k, C_{m_1} will be queued ahead of C_{m_2} in Q at p_k.

As C_{m_1} is enqueued ahead of C_{m_2} in Q at p_k, causal delivery of m_1 with respect to m_2 is guaranteed. □

Corollary 2. *Algorithm 1 guarantees strong safety.*

Corollary 3. *Algorithm 1 satisfies Definition 3 for causal order unicasting.*

Note that the broadcasts in lines 3 and 9 can be replaced by a multicast to a group of size k as long as the upper bound on the number of Byzantine processes in the group satisfies $k \geq 2t + 1$.

Algorithm 1 for unicasts can be adapted for multicast to groups, each group being identified by G_{id_m}, with straightforward modifications (such as replacing j by G_{id_m} in line 3 and replacing "$recipient = i$" by "$i \in G_{id_m}$" in line 6). This adaptation guarantees liveness and strong safety but does not provide the reliability properties (BCM-Validity, BCM-Termination-1, BCM-Agreement, BCM-Integrity). To satisfy these, one could replace the regular broadcast in line 3 by a Byzantine Reliable Broadcast primitive BR_broadcast. However, two messages sent by a process via BR_broadcast are not guaranteed to be delivered in the order they were sent (thus even FIFO order is not guaranteed) [20] or in a total order. Hence, we need to use a different approach for providing reliable causal multicast. A different approach that invokes FIFO-total order broadcast via BA_broadcast, for asynchronous systems in given in Sect. 6. This algorithm in Sect. 6 is a probabilistic algorithm because its BA_broadcast cannot be implemented in an asynchronous system deterministically.

6 Causal Order Multicast for an Asynchronous System

In Algorithm 2 we present a causal ordering algorithm guaranteeing strong safety and liveness in the presence of Byzantine processes for asynchronous systems. Algorithm 2 is a non-deterministic algorithm, complementing the result in [21,22]. Similar to Algorithm 1, Algorithm 2 requires that key generation and distribution has been accomplished by a trusted dealer prior to start of execution. Therefore, all processes have access to global PK (public key), VK (verification key) and have a local SK_i (secret key). In addition to this, all multicast groups share a unique symmetric key for encryption and decryption of messages intended for them. Algorithm 2 *double encrypts* each message, first with the group key (K_G) and then with the system key (PK) and invokes a source-order preserving atomic broadcast on the resulting ciphertext. Upon receiving the ciphertext, all processes compute their respective decryption shares and the recipients of the multicast message enqueue the ciphertext in their respective FIFO delivery queues and broadcast a request to the system for decryption shares. Upon receiving the required number of valid and unique decryption shares, the ciphertext is decrypted to obtain the ciphertext encrypted with the group key. When this ciphertext reaches the head of the delivery queue it is decrypted with the group key to obtain the original message and delivered to the application. The number of Byzantine failures that Algorithm 2 can tolerate is dependent on the tolerance of the atomic broadcast primitive used. The requirement for atomic broadcast is typically $n > 3t$.

Each message m has a globally unique identifier id_m assigned by the sender. In the Algorithm 2 pseudo-code, technically C_m, C'_m, σ_i^m, S should be C_{id_m},C'_{id_m}, $\sigma_i^{id_m}$, S_{id_m} respectively; however to simplify the presentation, we use the first version while keeping in mind that the data structure is to be associated with a particular id_m.

Lemma 1. *(***Process Order:***) If a process* BA_broadcast*s m_1 before it* BA_-broadcast*s m_2, i.e., $m_1 \rightarrow m_2$, and if some correct process is* BA_deliver*ed m_1 and m_2, then all correct processes are* BA_deliver*ed m_1 before m_2.*

Proof. Follows from the source-FIFO ordering property of BA_broadcast. □

Lemma 2. *(***Message Order:***) If a (correct or Byzantine) process* BA_broad-cast*s message m_2 after it* BA_deliver*s, decrypts, and dequeues m_1, i.e., $m_1 \rightarrow m_2$, then no correct process* BA_deliver*s m_2 before it* BA_deliver*s m_1.*

Proof. When a process p_x BA_delivers, decrypts, and dequeues m_1, it must have received a decryption share $\sigma_x^{m_1}$ from at least one correct process p_c which implies that at least one correct process p_c must have already BA_delivered m_1. By BAB-Agreement, all correct processes BA_deliver m_1. The correct process p_c will necessarily never have BA_delivered m_2 before it has BA_delivered m_1. From the BAB-Agreement property, if m_2 is BA_delivered to any correct process, it will necessarily be BA_delivered to all correct processes including p_c. At p_c, m_2 will be BA_delivered after m_1. Therefore by the BAB-Total-Ordering property, m_2 will be BA_delivered after m_1 at all correct processes. □

Algorithm 2: Asynchronous Secure Causal Multicast

Data: Each process has access to PK (global public key) and VK (global verification key) as well as a local secret key SK_i. Each process maintains a FIFO queue Q for incoming application messages. All processes in a multicast group G locally store the group key K_G.

```
1  when $p_i$ has to send $m$ to $G_{id_m}$ via BC_multicast($m, G_{id_m}$):
2  $C'_m = Enc(K_{G_{id_m}}, m)$
3  $C_m = E(PK, C'_m, id_m)$
4  BA_broadcast($C_m, G_{id_m}, id_m$)

5  when $\langle C_m, G_{id_m}, id_m \rangle$ arrives at $p_i$ via BA_deliver():
6  $\sigma_i^m = D(SK_i, C_m)$
7  if $p_i \in G_{id_m}$ then
8      $Q.push(C_m)$
9      $broadcast(request, id_m)$ to $\forall p_x$

10 when $p_i$ receives $\langle request, id_m \rangle$ from $p_j$:
11 while $C_m$ has not arrived at $p_i$ do
12     wait in a non-blocking manner
13 if $p_j \in G_{id_m}$ then
14     send($\sigma_i^m$) to $p_j$

15 when $p_i$ receives $(t + 1)$ valid $\langle \sigma_x^m \rangle$ messages:
16 Store $(t + 1)$ decryption shares in set $S$
17 $C'_m = C(VK, C_m, S)$
18 replace $C_m$ in $Q$ with $C'_m$

19 when the application is ready to process a message at $p_i$:
20 if $Q.head()$ has been decrypted using decryption shares then
21     $C'_m = Q.pop()$
22     $m = Dec(K_{G_{id_m}}, C'_m)$ using group key $K_{G_{id_m}}$
23     BC_deliver($m$)
```

Theorem 3. *Algorithm 2 guarantees BCM-Validity, BCM-Termination-1, BCM-Agreement, BCM-Integrity and BCM-Causal-Order in the presence of Byzantine processes.*

Proof. 1. (BCM-Validity:) An incoming message m at a correct process p_i sent to group G_{id_m} is enqueued, double-decrypted, dequeued and BC_delivered only if (i) p_i belongs to G_{id_m}, and (ii) the (double-encrypted) message was BA_delivered. This follows from the pseudo-code. As m was BA_delivered, by BAB-Validity, it must also have been BA_broadcast by $sender(m)$ with parameter G_{id_m}. Therefore, a message m can be BC_delivered at p_i only if it is BC_multicasted in lines 1–4 by $sender(m)$ to G_{id_m} via BA_broadcast after double-encryption and $p_i \in G_{id_m}$.

2. (BCM-Termination-1:) Consider message m BC_multicast by a correct process p_i to group G_{id_m}. p_i executes BA_broadcast(C_m, G_{id_m}, id_m) at line 4, ensuring via its properties of BAB-Termination-1 and BAB-Agreement that all correct processes receive C_m via BA_deliver and compute their respective decryption shares at lines 5–6. Once $p_j \in G_{id_m}$ receives C_m via BA_deliver, it pushes C_m into a FIFO queue at lines 5–9. p_j then broadcasts a request for decryption shares to all processes at line 9. p_j will receive decryption share σ_x^m from each correct process p_x eventually, once p_x has also BA_delivered C_m and computed its decryption share. Since there are $(2t+1)$ correct processes, p_j is guaranteed to receive the required $(t+1)$ decryption shares in line 15. Therefore, C_m is guaranteed to be decrypted and C'_m is guaranteed to be present in Q (lines 15–18). The encryption of m using the group key $K_{G_{id_m}}$ in line 2 and its corresponding decryption in line 22 ensures that only members of group G_{id_m} can access the content of C'_m.
   A ciphertext $C_{m'}$ present ahead of C'_m in Q at p_j may have been sent via BA_broadcast by a Byzantine process or by a correct process. Irrespective of this, as $C_{m'}$ has been BA_delivered to (correct) process p_j, by the BAB-Agreement property of BA_broadcast it $(C_{m'})$ would also have been BA_delivered to all at least $2t+1$ correct processes p_x which would compute their decryption share $\sigma_x^{m'}$ in line 6 and reply to p_j's request broadcast in line 9 with the decryption share $\sigma_x^{m'}$ in line 14. Thus p_j is guaranteed to get $(t+1)$ decryption shares and decrypt $C_{m'}$ to $C'_{m'}$ which can then get popped from Q after its double-decryption using its group key $K_{G_{id_{m'}}}$. This allows C'_m to be at *head*(Q) and get processed when popped (lines 19 to 23). Therefore, any message enqueued in the delivery queue will eventually reach the head of the queue. This means m is guaranteed to reach the head of Q and eventually be BC_delivered in lines 19–23 ensuring BCM-Termination-1.
3. (BCM-Agreement:) If a correct process $p_i \in G_{id_m}$ BC_delivers a message m, it means that C_m was BA_delivered in lines 5–9. By the BAB-Agreement property of BA_broadcast, this means that all correct processes in the system BA_deliver C_m, compute their respective decryption shares and push C_m in their respective delivery queues if they are part of G_{id_m}. Therefore, if there exists another correct process $p_j \in G_{id_m}$, it will receive C_m via BA_delivery and insert C_m in its delivery queue. From the reasoning for BCM-Termination-1 given in the above item, we know that any message enqueued in the delivery queue eventually reaches the head of the queue. Therefore, C_m will eventually reach the head of the queue. Additionally, between lines 10–18, p_j will receive $(t+1)$ decryption shares required to decrypt C_m in the queue since there are $(2t+1)$ correct processes in the system. Therefore, m is guaranteed to be BC_delivered at p_i and any correct p_j in G_{id_m} in lines 19–23, thereby guaranteeing BCM-Agreement.
4. (BCM-Integrity:) By the BAB-Integrity property of BA_broadcast a message is BA_delivered at most once at a correct process. Therefore any incoming message will be enqueued (lines 5–9) and dequeued from the delivery queue

(lines 19–23) at most once at a correct process. Hence, any given message m will be BC_delivered at most once at a correct process.

5. (BCM-Causal-Order:) Consider multicast messages m and m' sent by p_x and p_y to groups containing a correct process p_k, where $m \rightarrow m'$. Then there must exist a message chain $\langle m_0, m_1, m_2, \ldots m_{z-1}, m_z = m' \rangle$ such that (i) m_0 was sent (via BA_broadcast) by $p_{i_0} = p_x$ after it sent m (via BA_broadcast), (ii) m_{a-1} for $a \in [1, z]$ was BA_delivered, decrypted, and dequeued (BC_delivered) by p_{i_a} before p_{i_a} sent m_a (by executing BA_broadcast), and (iii) $p_{i_z} = p_y$.
Let p_k BA_deliver m and m'. Further, all correct processes BA_deliver m_{a-1} and m_a, for $a \in [1, z]$. By Lemma 1 (*Process Order*), $m \rightarrow m_0$ and all correct processes will BA_deliver m_0 after m. By Lemma 2 (*Message Order*), $m_{b-1} \rightarrow m_b$, for $b \in [1, z]$, hence all correct processes will BA_deliver m_{b-1} before m_b. Hence by transitivity, it follows that all correct processes will BA_deliver m before $m_z = m'$. As p_k is a common member of multicast groups addressed by m and m', it will enqueue m, i.e., C_m, before m', i.e., $C_{m'}$ in Q. This ensures that m will be dequeued and delivered before m', thus satisfying BCM-Causal-Order.

□

Corollary 4. *Algorithm 2 guarantees Byzantine-tolerant causal order multicast as per Definition 11.*

Since Algorithm 2 guarantees BCM-Termination-1 and BCM-Agreement, it implicitly guarantees liveness. Algorithm 2 explicitly guarantees Strong Safety because it guarantees BCM-Causal-Order.

Corollary 5. *Algorithm 2 satisfies Definition 3 for causal order multicasting.*

6.1 Adaptations to Special Cases

Asynchronous System, Unicast: The encryption in line 2 and corresponding decryption in line 22 are done using the symmetric key K_{ij} when p_i is sending to p_j. In line 4, the second parameter of BA_broadcast is j and in line 5, the second parameter of the delivered message is *recipient*. Line 7 tests if $p_i = recipient$. Line 13 tests if p_j is the recipient of message m.

Asynchronous System, Broadcast: Lines 2 and 22 can be deleted as the group contains all processes and there is no need to encrypt with the group key.

Synchronous System; Multicast, Unicast, and Broadcast: Algorithm 2 directly applies to a multicast in a synchronous system. The difference is that the BA_broadcast which is necessarily a probabilistic algorithm in the asynchronous system now becomes a deterministic algorithm. The special cases of unicast and broadcast in an asynchronous system likewise work in a synchronous system with the probabilistic BA_broadcast now becoming a deterministic BA_broadcast. Due to the high message complexity and latency of this version of unicast in a synchronous system, Algorithm 1 is more efficient for unicast.

7 Discussion

We conjecture that it is impossible to provide strong safety in Byzantine-tolerant causal order for multicasts, (unicasts, or broadcasts) in synchronous systems without using cryptographic techniques, complementing the impossibility result [21,22] for asynchronous systems. This is because in isolation, a Byzantine process is free to delete true dependencies of its messages on messages that it sends out. By using cryptographic techniques, this advantage is nullified by making the Byzantine process dependent on correct processes to decipher and read incoming messages. This makes sure that a Byzantine process cannot falsify/delete causal dependencies because it no longer operates in isolation and requires cooperation of one or more correct processes in reading and sending messages.

In this paper, we have extended previous work that provided weak safety of causal order unicasts/multicasts to now provide strong safety with the use of threshold encryption for both synchronous and asynchronous systems. The causal ordering algorithm for asynchronous systems is non-deterministic, while the algorithm for synchronous systems is deterministic. The synchronous algorithm for unicasts (Algorithm 1) has a low cost with message complexity $O(n)$ point-to-point messages per application message, but assumes assistance from the network in terms of an upper bound on message latency. The asynchronous algorithm for multicasts (Algorithm 2) has a higher message cost of at least $O(n^2)$ (depending on the implementation of the BA_broadcast primitive [7,12,16]) plus $O(n \cdot |G|)$ point-to-point messages per multicast to group G, but does not assume any support from the network. Depending on the application requirements and constraints, either of the two algorithms can be used for causal ordering.

References

1. Auvolat, A., Frey, D., Raynal, M., Taïani, F.: Byzantine-tolerant causal broadcast. Theoret. Comput. Sci. **885**, 55–68 (2021)
2. Birman, K.P., Joseph, T.A.: Reliable communication in the presence of failures. ACM Trans. Comput. Syst. (TOCS) **5**(1), 47–76 (1987)
3. Bracha, G.: Asynchronous byzantine agreement protocols. Inf. Comput. **75**(2), 130–143 (1987)
4. Bracha, G., Toueg, S.: Asynchronous consensus and broadcast protocols. J. ACM (JACM) **32**(4), 824–840 (1985)
5. Cachin, C., Kursawe, K., Petzold, F., Shoup, V.: Secure and efficient asynchronous broadcast protocols. IACR Cryptol. ePrint Arch, p. 6 (2001)
6. Chandra, T.D., Toueg, S.: Unreliable failure detectors for reliable distributed systems. J. (JACM) **43**(2), 225–267 (1996)
7. Correia, M., Neves, N.F., Veríssimo, P.: From consensus to atomic broadcast: time-free byzantine-resistant protocols without signatures. Comput. J. **49**(1), 82–96 (2006)
8. Défago, X., Schiper, A., Urbán, P.: Total order broadcast and multicast algorithms: taxonomy and survey. ACM Comput. Surv. **36**(4), 372–421 (2004)

9. Duan, S., Reiter, M.K., Zhang, H.: Secure causal atomic broadcast, revisited. In: 2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 61–72. IEEE (2017)
10. Dwork, C., Lynch, N.A., Stockmeyer, L.J.: Consensus in the presence of partial synchrony. J. ACM **35**(2), 288–323 (1988)
11. Fischer, M.J., Lynch, N.A., Paterson, M.: Impossibility of distributed consensus with one faulty process. J. ACM **32**(2), 374–382 (1985)
12. Gągol, A., Leśniak, D., Straszak, D., Świętek, M.: Aleph: efficient atomic broadcast in asynchronous networks with byzantine nodes. In: Proceedings of the 1st ACM Conference on Advances in Financial Technologies, pp. 214–228 (2019)
13. Hadzilacos, V., Toueg, S.: A modular approach to fault-tolerant broadcasts and related problems. Technical report 94–1425, p. 83. Cornell University (1994)
14. Kshemkalyani, A.D., Singhal, M.: Necessary and sufficient conditions on information for causal message ordering and their optimal implementation. Distributed Comput. **11**(2), 91–111 (1998)
15. Kshemkalyani, A.D., Singhal, M.: Distributed Computing: Principles, Algorithms, and Systems. Cambridge University Press, Cambridge (2011)
16. Kursawe, K., Shoup, V.: Optimistic asynchronous atomic broadcast. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 204–215. Springer, Heidelberg (2005). https://doi.org/10.1007/11523468_17
17. Lamport, L., Shostak, R.E., Pease, M.C.: The byzantine generals problem. ACM Trans. Program. Lang. Syst. **4**(3), 382–401 (1982)
18. Milosevic, Z., Hutle, M., Schiper, A.: On the reduction of atomic broadcast to consensus with byzantine faults. In: 2011 IEEE 30th International Symposium on Reliable Distributed Systems, pp. 235–244. IEEE (2011)
19. Misra, A., Kshemkalyani, A.D.: Causal ordering in the presence of byzantine processes. In: 28th IEEE International Conference on Parallel and Distributed Systems, ICPADS, pp. 130–138. IEEE (2022)
20. Misra, A., Kshemkalyani, A.D.: Causal ordering properties of byzantine reliable broadcast primitives. In: Colajanni, M., Ferretti, L., Pardal, M.L., Avresky, D.R. (eds.) 21st IEEE International Symposium on Network Computing and Applications, NCA 2022, pp. 115–122. IEEE (2022)
21. Misra, A., Kshemkalyani, A.D.: Solvability of byzantine fault-tolerant causal ordering problems. In: Koulali, M., Mezini, M. (eds.) NETYS 2022. LNCS, vol. 13464, pp. 87–103. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-17436-0_7
22. Misra, A., Kshemkalyani, A.D.: Byzantine fault-tolerant causal ordering. In: 24th International Conference on Distributed Computing and Networking, ICDCN 2023, Kharagpur, India, January 4–7, 2023, pp. 100–109. ACM (2023)
23. Pease, M.C., Shostak, R.E., Lamport, L.: Reaching agreement in the presence of faults. J. ACM **27**(2), 228–234 (1980)
24. Raynal, M.: Fault-Tolerant Message-Passing Distributed Systems: An Algorithmic Approach. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-94141-7
25. Shoup, V., Gennaro, R.: Securing threshold cryptosystems against chosen ciphertext attack. J. Cryptol. **15**(2), 75–96 (2002)