Synchronous Byzantine Lattice Agreement in O(log(f)) Rounds

Giuseppe Antonio Di Luna, Emmanuelle Anceaume, Silvia Bonomi, Leonardo Querzoni

Synchronous Byzantine Lattice Agreement in $\mathcal{O}(\log(f))$ Rounds

Giuseppe Antonio Di Luna *, Emmanuelle Anceaume †, Silvia Bonomi * and Leonardo Querzoni *
* DIAG, Sapienza University of Rome — Email: {*diluna,bonomi,querzoni*}*@diag.uniroma1.it*
† CNRS, Univ. Rennes, Inria, IRISA — Email: *emmanuelle.anceaume@irisa.fr*

***Abstract*—In the Lattice Agreement (LA) problem, originally proposed by Attiya et al. [1], a set of processes has to decide on a chain of a lattice. More precisely, each correct process proposes an element e of a certain join-semi lattice L and it has to decide on a value that contains e. Moreover, any pair p_i, p_j of correct processes has to decide two values dec_i and dec_j that are comparable (e.g., $dec_i \leq dec_j$ or $dec_j < dec_i$).**

In this paper we present new contributions for the synchronous case. We investigate the problem in the usual message passing model for a system of n processes with distinct unique IDs. We first prove that, when only authenticated channels are available, the problem cannot be solved if $f = n/3$ or more processes are Byzantine. We then propose a novel algorithm that works in a synchronous system model with signatures (i.e., the *authenticated message* model), tolerates up to f byzantine failures (where $f < n/3$) and that terminates in $\mathcal{O}(\log f)$ rounds. We discuss how to remove authenticated messages at the price of algorithm resiliency ($f < n/4$). Finally, we present a transformer that converts any synchronous LA algorithm to an algorithm for synchronous Generalised Lattice Agreement.

***Index Terms*—Lattice Agreement, Byzantine Fault Tolerance**

I. Introduction

Fault-tolerance is a key research topic in distributed computing: reliable algorithms have been deeply investigated in classic message passing [2] and in newer model of computations [3], [4], [5]. A special mention has to be given to algorithms for distributed agreement [2]. They represent a cornerstone of todays cloud-based services. In particular, practical and efficient implementations of distributed consensus, transformed Internet from a large computers network to a world-scale service platform. Despite its fundamental role, distributed consensus is impossible to solve deterministically in asynchronous settings, where communication latencies cannot be bounded. To cope with this limit, practical systems trade off consistency criteria (allowing weaker *agreement* properties) with liveness (guaranteeing termination only in long-enough grace periods where the system "behaves" like a synchronous one).

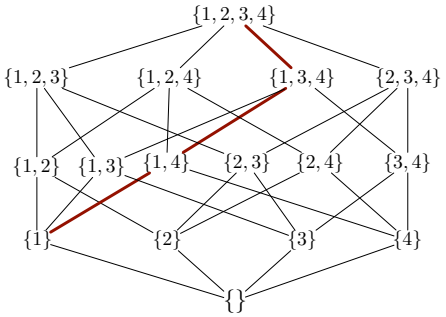

Fig. 1. Hasse diagram of the semilattice induced over the power set of $\{1,2,3,4\}$ using the union operation as join. Given two elements e, e' of the lattice if $e < e'$, then there is an "upward" path connecting e to e' (e.g., $\{1\} \leq \{1,3,4\}$, but $\{2\} \not\leq \{3\}$). Any two elements e, e' of the semilattice have a join $e \oplus e' = e \cup e'$ and $e \oplus e' \geq e, e'$ (e.g., $\{1\} \oplus \{2,3\} = \{1,2,3\}$). The red edges indicate a possible chain (i.e., sequence of increasing values).

For such a reason, agreement properties weaker than consensus proved to be extremely effective for the implementation of a broad family of distributed applications, since they can be used in systems where consensus cannot be solved, or they can be faster than consensus algorithms circumventing time-complexity lower bounds.

Lattice Agreement. In this paper we investigate an agreement problem that is weaker than consensus: the Lattice Agreement (LA) problem. In LA, introduced by Attiya et al. [1], each process p_i has an input value x_i drawn from the join semilattice and must decide an output value y_i, such that *(i)* y_i is the join of x_i and some set of input values and *(ii)* all output values are comparable to each other in the lattice, that is they are all located on a single chain in the lattice (see Figure 1). LA describes situations in which processes need to obtain some knowledge on the global execution of the system, for example a global photography of the system. In particular Attiya et al. [1] have shown that in the asynchronous shared memory computational model, implementing a snapshot object is equivalent to solving the Lattice Agreement problem. Faleiro et al. [8] have shown that in a message passing system a majority of correct processes and reliable communication channels

TABLE I
COMPARISON OF ALGORITHMS FOR BYZANTINE LATTICE AGREEMENT WHEN THE LATTICE HAS AN HEIGHT THAT IS GREATER THAN f.

Model	*Assumption*	*Paper*	*Resiliency*	*Time*	*Messages*
(SYNC)	Auth. Links (No Sign.)	This paper	$f < \lceil n/4 \rceil$	$\mathcal{O}(\log(f))$	$\mathcal{O}(n^2 \log(f))$
		Zheng et al. [6]	$f < \lceil n/3 \rceil$	$\mathcal{O}(\sqrt{f})$	$\mathcal{O}(n^2\sqrt{f})$
	Auth. Messages (Signatures)	This paper	$f < \lceil n/3 \rceil$	$\mathcal{O}(\log(f))$	$\mathcal{O}(n^2 \log(f))$
(ASYNC)	Auth. Links (No Sign.)	Di Luna et al. [7]	$f < \lceil n/3 \rceil$	$\mathcal{O}(f)$	$\mathcal{O}(n^2)$
	Auth. Messages (Signatures)	Di Luna et al. [7]	$f < \lceil n/3 \rceil$	$\mathcal{O}(f)$	$\mathcal{O}(f \cdot n)$

are sufficient to solve LA in asynchronous systems, proposing a Replicated State Machine with commutative updates built on top of a generalized variant of their LA algorithm. Generalized Lattice Agreement (GLA) is a version of LA where processes propose and decide on a, possibly infinite, sequence of values. The restriction of having only commutative updates is justified by the possibility of developing faster algorithms. It is well known [9] that consensus cannot be solved in synchronous systems in less than $\mathcal{O}(f)$ rounds, even when only crash failures are considered, on the other hand it has been shown [10] that, when crash failures are considered, Lattice Agreement can be solved in $\mathcal{O}(\log f)$ rounds[1].

Byzantine Failures. All these papers consider process failures, but assume that such failures are non-malicious. More recently, some works started proposing LA algorithms that tolerate Byzantine failures. The first one has been by Nowak and Rybicki [11]: they introduced LA with Byzantine failures and proposed a definition of Byzantine LA in which decisions of correct processes are not allowed to contain values proposed by Byzantine processes. Using such definition, authors have shown that the number of correct processes needed to solve LA depends on the structure of the lattice, and they have proposed a LA algorithm for specific kinds of lattices.

Successively, Di Luna et al. [7] proposed a less restrictive definition of Byzantine LA, in which correct processes can decide also values proposed by Byzantine. Authors have then shown that LA can be solved for any possible lattice when $f < \frac{n}{3}$: they proposed a solution for Byzantine LA in asynchronous systems that terminates in $\mathcal{O}(f)$ rounds; the same paper also proposed a Generalized version of the algorithm and built on top of it a Replicated State Machine that executes commutative operations. In this paper we adopt the Byzantine LA definition from [7], since it allows to circumvent some restrictions of [11] and it is usable in many practical scenario. The same definition has also been recently used by Zheng and Garg [6], where they show that LA can be solved in synchronous systems with $\mathcal{O}(\sqrt{f})$ rounds also in presence of Byzantine failures.

[1]Actually, it can be solved faster than $\log(f)$ on specific lattices, the ones having height less than $\log f$ [10], however in this paper we are interested only in worst case performance.

Contributions. In this paper we present new contributions for the Byzantine LA problem in synchronous settings. Our first results is for systems with only authenticated channels (i.e., signatures are not available), in such systems we show that Byzantine LA on arbitrary lattices cannot be solved, in synchronous systems, with $f = \lceil n/3 \rceil$ or more faulty processes (Section III). Interestingly, such proof shows that the algorithm of Zheng and Garg [6] is tight in the number of tolerable failures. On the positive side we show algorithms that solve LA and Generalized LA, with and without signatures, having better running time that the state-of-the-art. Looking at the model with signatures, we show a novel algorithm for LA that works in a synchronous system model, tolerates up to f byzantine failures (where $f < \lceil n/3 \rceil$) and that terminates in $\mathcal{O}(\log f)$ rounds. The algorithm improves over the LA_β algorithm from Garg at al. [10] by using a similar construction, but adding tolerance to Byzantine failures. We make use of a modified Gradecast algorithm that allows processes to prove that a message has been seen by all correct processes in the system. (Sections IV-V) We conclude our investigation on LA by briefly discussing how to remove signatures and make our construction work only with authenticated channels trading-off part of its resiliency: we are able to tolerate $f < \lceil n/4 \rceil$ failures (Section VI). In the last part of the manuscript, we devote our attention to Generalized Lattice Agreement (Section VII). Specifically, we show a transformer that, using as building block a generic LA algorithm, creates a Generalized Lattice Agreement algorithm. To the best of our knowledge this is the first time GLA is investigated in synchronous systems. Table I compares our results with the literature. For space reason some details and proofs are omitted and can be found in the full version [12].

II. SYSTEM MODEL, NOTATION, AND PRELIMINARIES

We use the usual message passing models with unique identifiers (IDs). There is a set Π of n processes with unique IDs in $\{1, \ldots, n\}$ connected by a complete

communication graph. The system is synchronous, and the execution of the algorithm can be divided in discrete finite time units called rounds. In each round a process is able to send messages to its neighbours (*send phase*), and receive all messages sent to it at the beginning of the round (*receive phase*). Processes in Π are partitioned in two sets F and C. Processes in C are correct, they faithfully follow the distributed protocol. Processes in F are Byzantine, they arbitrarily deviate from the protocol. As usual when Byzantine failures are considered, we assume that the communication channels are authenticated by mean of Message Authentication Codes (MAC). The *authenticated channels* are the only assumption used in Section III. In Section IV we assume that there is a public key infrastructure that allows processes to cryptographically sign messages, that can be lately verified by other processes. This model has *authenticated messages*. Byzantine processes are polynomially bounded and cannot forge signatures of correct processes. For an easier presentation we explain our algorithms for the case of $n = 3f + 1$, where $f = |F|$, however they can be easily adapted for any other $n > 3f + 1$.

Notation. With ϵ we indicate the empty string. Given a string G, with $|G|$ we indicate the length of the string ($|\epsilon| = 0$), with $G[j]$ and $0 \leq j < |G|$ we indicate the character of string G in position j. With $G[k : l]$ (given $0 \leq k \leq l \leq |G|$), we indicate the substring of G between position k and l. As an example given $G = ssms$, we have $G[0] = s$ and $G[0 : 1] = ss$. Given two strings a and b with $a \cdot b$ we indicate the string obtained by concatenating b after a.

a) The Byzantine Lattice Agreement Problem: Each process $p_i \in C$ starts with an initial input value $pro_i \in E$ with $E \subseteq V$ (set E is a set of allowed proposal values). Values in V form a join semi-lattice $L = (V, \oplus)$ for some commutative join operation $\oplus$: for each $u, v \in V$ we have $u \leq v$ if and only if $v = u \oplus v$. Given $V' = \{v_1, v_2, \ldots, v_k\} \subseteq V$ we have $\bigoplus V' = v_1 \oplus v_2 \oplus \ldots \oplus v_k$.

The task that processes in C want to solve is the one of Lattice Agreement, and it is formalised by the following properties:

- **Liveness:** Each process $p_i \in C$ eventually outputs a decision value $dec_i \in V$;
- **Stability:** Each process $p_i \in C$ outputs a unique decision value $dec_i \in V$;
- **Comparability:** Given any two pairs $p_i, p_j \in C$ we have that either $dec_i \leq dec_j$ or $dec_j \leq dec_i$;
- **Inclusivity:** Given any correct process $p_i \in C$ we have that $pro_i \leq dec_i$;
- **Non-Triviality:** Given any correct process $p_i \in C$ we have that $dec_i \leq \bigoplus(X \cup B)$, where X is the set of proposed values of all correct processes ($X : \{pro_i|$ with $p_i \in C\}$), and $B \subseteq E$ is $|B| \leq f$.

Lattice definitions. A path of length k between two distinct elements u and $v$of the latttice is a sequence of $k + 1$ distinct elements $(e_0, e_2, \ldots, e_k)$ such that $e_0 = u \leq e_1 \leq e_2 \leq \ldots \leq e_{k-1} \leq e_k = v$. As an example the path between $\{1, 2, 3\}$ and $\{1\}$ in the lattice of Figure 1 has length 2. We say that a $v \in V$ is minimal if it does not exists $u \in V$, with $u \neq v$, such that $u \oplus v = v$ (i.e., it does not exists an $u \leq v$). As in [6] we define the height of an element v in a lattice $(V, \oplus)$ has the length of the longest path from any minimal element to v in the lattice (as an example the lattice in Figure 1 has height 4). A sub-lattice of $(V, \oplus)$ is a subset U of V closed with respect to the join operation, the definition of height for a sub-lattice does not change.

Preliminaries. In the rest of the paper we will assume that L is a semi-lattice over sets (V is a set of sets) and $\oplus$ is the set union operation. This is not restrictive, it is well known that any join semi-lattice is isomorphic to a semi-lattice of sets with set union as join operation. An important lattice is the one on the power set of the first $\{1, \ldots, n\}$ natural numbers with the union as join operation (see Figure 1), we will use as shorthand for such lattice the notation L_n (note that n is also the number of processes). We will show that an algorithm solving lattice agreement exclusively on such a lattice (the GAC of Section IV) can be used as building block to solve lattice agreement on an arbitrary lattice (Section V). When L_n is considered, the height of an element e is equal to its cardinality (i.e, $|e|$ see Figure 1), given a sub lattice of L_n its height is upper bounded by the difference between the minimum and maximum cardinality of its elements.

III. Authenticated Channels and No Signature - Necessity of $3f + 1$ processes in synchronous systems

In the following we show that $3f + 1$ processes are necessary when there are no signatures.

Lemma 1. *It does not exist any algorithm solving Byzantine LA on arbitrary lattices in a synchronous system with 3 processes when one is faulty. The impossibility holds even when relaxing the Non-Triviality allowing $|B| \leq k$ for any fixed k.*

Proof. We first discuss the case of $k = 1$. Let $\mathcal{A}$ be an algorithm solving LA with 3 processes when one is faulty. Since $\mathcal{A}$ works on arbitrary lattices it should

also work on the lattices induced by the union operation on the power set of the first 6 natural numbers with $E = \{\{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}\}$. Now let us consider the hexagonal system of Figure 2. Such a system is constituted by 6 processes $p_1, p_2, p_3, p_4, p_5, p_6$ with an edge between each p_i, p_j such that $i = j \pm 1$ and one edge between p_1 and p_6. Each of the six processes has as input an unique value in $[1, 6]$, just for simplicity process p_i has input $\{i\}$. Note that even if $\mathcal{A}$ is an algorithm for three processes, it is possible to execute $\mathcal{A}$ on the hexagon, but its behaviour does not necessarily follows the LA specification.

In the figure we have 6 triangles, each triangle is related to a corresponding edge in the hexagon. The relationship is such that the view of two neighbour processes in the hexagon is equal to the view of two processes in a triangle where the third process is a Byzantine simulating the behaviour of the other processes in the hexagon. As example: the view of processes p_1, p_2 in the hexagon is the same that p_1, p_2 would have in triangle t_{green}, analogously the view of p_6, p_1 in the hexagon is the same view of p_6, p_1 in t_{blue}. Note that $\mathcal{A}$ once executed on any of the triangle in the figure has to follow the LA specification.

A run of $\mathcal{A}$ on the hexagon in principle has an undefined behaviour. However we observe that a run of $\mathcal{A}$ on the hexagon eventually terminates on each process, this is because each process has a local view that is consistent with a system of 3 processes one of which is a Byzantine. Recall, that the local view of each process p_i in the hexagon is exactly the same view that the process has in the two triangles on the right, and $\mathcal{A}$ being a correct algorithm when 3 processes are considered the algorithm will correctly terminate in each triangle in the right.

Moreover, each process will output a decision value that must be the same that the process will output in the corresponding triangles.

Let $dec_1, dec_2, dec_3, dec_4, dec_5, dec_6$ be the decisions of processes dictated by $\mathcal{A}$ (naturally we have dec_i decision of p_i). The triangles on the right impose a certain number of comparability relationships among these decisions. Recall, that each decision is a subset, not necessarily proper, of $[1, 6]$, and that the comparability in this setting is the relationship of inclusion. An example is triangle t_{green} that imposes the comparability between dec_1 and dec_2, that is either $dec_1 \subset dec_2$ or $dec_2 \subseteq dec_1$. In the following we use $dec_i \leftrightarrow dec_j$ to indicate that dec_i must be comparable with dec_j. Before continuing with the proof we give the following technical observation: consider a collection of m sets $S_1, S_2, \ldots, S_m$ such that for each S_i we have $i \in S_i$. If it holds that $S_j \leftrightarrow S_{i+1}$ for all $j \in [1, m-1]$ then there exists an $|S_k| \geq m$.

Therefore, let us take w.l.o.g. dec_1, and let us walk in clockwise direction for 3 steps on the hexagon. On this walk we have: $dec_1 \leftrightarrow dec_2 \leftrightarrow dec_3 \leftrightarrow dec_4$, by the inclusivity property we have for each dec_i that $i \in dec_i$. We can apply the aforementioned observation and state that one of the dec_i has cardinality at least 4 violating the non-triviality property of $\mathcal{A}$ on some triangle (recall that k being equal to 1 we have $|B| \leq 1$). The generalised proof for an arbitrary k follows the same reasoning using: a lattice on the power set of the first $3(k+1)$ natural numbers, $E = \{\{1\}, \ldots, \{3(k+1)\}\}$ and a $3(k+1)-$gon instead of an hexagon. We then walk on the $3(k+1)-$gon for $k+2$ steps instead of 3. We will have a chain $dec_1 \leftrightarrow dec_2 \leftrightarrow \ldots \leftrightarrow dec_{k+3}$ where by inclusivity of $\mathcal{A}$ on the corresponding triangle we have $i \in dec_i$, and where our observation shows that one of the decisions contains at least $k+3$ distinct elements of E violating the non-triviality on some triangle. □

From the above lemma, by using a classic simulation argument we have:

Theorem 1. *It does not exist any algorithm solving Byzantine LA on arbitrary lattices in a synchronous system with $3f$ processes and f faulty if $|B| \leq f$. The same holds if $n < 3f$*

The proof of Theorem 1 does not work in a system with authenticated messages (i.e., signatures), it is therefore unkown whether $3f + 1$ processes are necessary also in this model. Interestingly, in [7] it is shown that, when the system is asynchronous, $3f + 1$ processes are necessary also when authenticated messages are available.

IV. Algorithm for L_n: Grade And Classify (GAC)

In this section we show an algorithm that works on L_n. The algorithm terminates in $\mathcal{O}(\log(n))$ rounds. We will then discuss in Section V how to use this algorithm so solve LA on arbitrary join semi-lattices, and how to adapt it to work in $\mathcal{O}(\log(f))$ rounds.

Our algorithm is based on the algorithmic framework of Zheng et al. [10] adapted to tolerate Byzantine failures. As in the original, the algorithm works by continuously partitioning processes in *masters* and *slaves* sets. Partitioning is recursively operated in successive epochs. Processes that have been assigned to the same

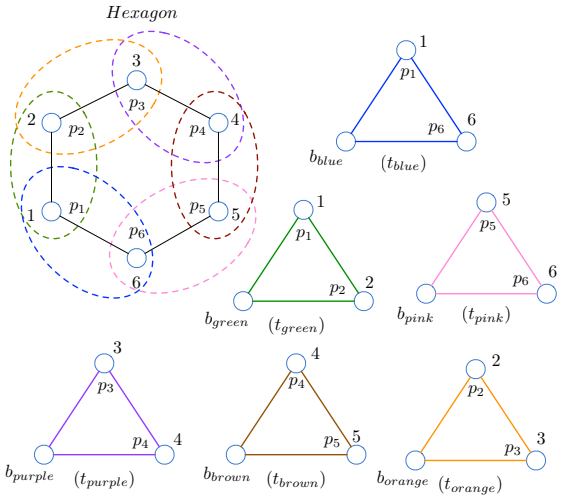

Fig. 2. Starting configuration of the Hexagon. For each edge there is a corresponding triangle that is a legal starting configuration of $\mathcal{A}$ with one Byzantine. As an example, take p_1, p_2 in the hexagon. They have the exactly same view of p_1, p_2 in t_{green} where the other node is b_{green} a Byzantine that simulates the behaviour of p_3, p_4, p_5, p_6 in the hexagon.

partition in each epoch are a "group". We indicate a generic group G at epoch ep with the string $s \cdot \sigma$, where σ is in $\{s, m\}^{ep-1}$. As an example, the string ssm indicates the set of process that at the end of epochs 0 and 1 entered the group of slaves (string ss) and then, at the end of epoch 2, entered in the masters group (string ssm). When we write $p \in G$, we indicate that process p belongs to the group of processes identified by string G.

The algorithm then enforces some properties on the partitions generated at an epoch ep on a Group G:

- each master process in $G \cdot m$ adjusts its proposal to be a superset of each possible decision of a slave process in $G \cdot s$.
- the height of a sub-lattice in which processes in $G \cdot m$ (or $G \cdot s$) are allowed to decide halves at each epoch.

Thanks to the above properties each group becomes independent and has to solve the lattice agreement on a lattice that has half of the height of the original. A key concept for our algorithm is the one of *"admissible value"*, a value is admissible, for a certain epoch, if it is ensured that it will not conflict with decisions of processes that have been elected as masters in a previous epoch. This is done by showing a cryptographical proof that such a value can be accepted by a slave since it is in the proposal value of each master the slave could conflict with. After $\mathcal{O}(\log(n))$ rounds the algorithm terminates (each group operates on a lattice constituted by a single point).

A. The Provable Gradecast Primitive

The algorithm makes use of the gradecast primitive introduced by Ben-Or et al. in [13]. Such primitive is similar to a broadcast, we have a sender process p_i that sends a message m, each other process p_j, after 3 rounds, outputs a tuple (p_i, m_j, c_j) where $c_j \in \{0, 1, 2\}$ is a score of the correctness of p_i. The gradecast ensures the following properties:

- for any two correct processes p_j, p_ℓ if $c_j > 0$ and $c_\ell > 0$ than $m_j = m_\ell$.
- for any two correct processes p_j, p_ℓ we have $|c_j - c_\ell| \leq 1$
- if the gradecast sender is correct, than for any correct process p_j we have $c_j = 2$.
- for any correct process p_j if $(p_i, m_j, 0)$ then $m_j = \perp$.

Intuitively, if we let processes communicate by mean of the gradecast primitive we force Byzantines to send at most two different messages to the set C of correct processes, and one of these messages has to be $\perp$. We modify the original gradecast to make it "provable". In our version of the gradecast each correct process outputs a tuple composed by 4 objects $(p_i, m_j, c_i, S_{i,m_j})$ where S_{i,m_j} is a special object that can either be $\perp$ or a *seen-all* proof. In case S_{i,m_j} is different from $\perp$, then it is a cryptographic proof that can be shown to other processes and it implies that, any correct process $p \in C$ has seen a rank, for the gradecast of message m_j from process p_i, that is at least 1. Moreover, we have that if $c_i = 2$ then $S_{i,m_j} \neq \perp$. Practically, the modification of the original gradecast are contained, and are limited to the second and third round of the algorithm. The original gradecast, with source p_s works as follows: in the first round p_s broadcasts a message m to all processes; in the second round each correct process relays the message received by p_s (it ignores messages from other processes); at the end of the second round a correct process selects the most frequent message received, and if such message was received by at least a quorum of $n - f$ processes then it relays the message at the beginning of the third round; at the end of the third round each correct selects the most frequent message received and it ranks it 2 and delivers it if the message was received by at least $n - f$ processes; if it was received by at least $f + 1$ it delivers it and ranks it 1, otherwise it delivers $\perp$ with rank 0.

In our version, see Algorithm 1, the relaying process signs the relay sent at the beginning of round 3 (see line 15) and a process that sees a message with rank 2 collects the $n - f$ (i.e., $2f + 1$ if $n = 3f + 1$) signed messages (see line 22). These signed messages constitute a proof that the message has been seen by all: delivered by each

correct with rank at least 1. Note that the algorithm is for a single instance and a single determined sender, however one can trivially run in parallel an instance for each possible sender in the system.

We do not prove the properties discussed above for our version of gradecast, they immediately derives from the correctness of the original algorithm [13].

Algorithm 1 *Provable GradeCast (PGC)* - Algorithm for sender process p_s and receiver process p_i

SND phase of round 1
if $p_i = p_s$ **then**
    BROADCAST(m) ▷ Executed only by sender p_s
▷ Code executed by any correct process $p_i \in \Pi$
RCV phase of round 1
$rcv = receive\_messages()$
m=select one message from p_s in rcv
SND phase of round 2
BROADCAST(m)
RCV phase of round 2
$rcv = receive\_messages()$
m = select the message in rcv received from the largest set of distinct sources.
SND phase of round 3
if m has been received from at least $n - f$ sources **then**
    BROADCAST$(m, sign(m))$ ▷ p_i relays the most frequent message signed by himself
RCV phase of round 3
$rcv = receive\_messages()$
remove from rcv all messages that are not correctly signed
m_i = select the message in rcv received from the largest set of distinct sources..
if m_i has been received by at least $n - f$ sources **then**
    $c_i = 2$
    S_{p_s,m_i} = select $n - f$ correctly signed messages in rcv for message m_i
else if m_i has been received by at least $f + 1$ sources **then**
    $c_i = 1, S_{p_s,m_i} = \perp$
else
    $m_i = \perp, c_i = 0, S_{p_s,m_i} = \perp$
return $< m_i, c_i, S_{p_s,m_i} >$

Observation 1. *Consider an instance of gradecast with source p_s. If a process p_i, whether Byzantine or correct, can produce a* seen-all *proof for a message m, then each process $p \in C$ delivered message m with rank at least 1 at the end of the gradecast instance.*

B. Detailed Algorithm Description

Processes communicate by provable gradecasts. The three rounds necessary to execute a gradecast instance form a single epoch. We assume that in each epoch there are n concurrent instances of gradecast running, one for each possible sender. The pseudo-code is in Algorithm 2.

1) Epoch $ep = 0$: Epoch 0 has a special structure. Correct processes belong to a single group $G = \epsilon$. In this

Algorithm 2 *GAC* - Algorithm for process p_i

$t_d = 0, t_u = n, t_m = \lfloor \frac{t_u}{2} \rfloor, G = \epsilon, Proofs = \{\}, pro_i = \emptyset$
function LA-PROPOSE(pro_i)
    $pro_i = pro_i$
    GRADECAST$(M = (pro_i, G, \perp))$ ▷ Epoch 0 - start
    P_i =RCV()
    $G = G \cdot s$
    $updateproofs(pro_i, P_i, 0)$ ▷ Epoch 0 - end
    for $ep \in [1, \ldots, \log(n)) + 1]$ **do**
        GRADECAST$(M = (pro_i, G, Proofs))$
        P_i =RCV()
        V_i =FILTER(P_i)
        if CLASSIFY$(V_i) = s$ **then**
            $G = G \cdot s$
            $t_d = t_d, t_u = t_m, t_m = t_d + \lfloor \frac{t_u - t_d}{2} \rfloor$
            $updateproofs(pro_i, P_i, ep)$
        else
            $G = G \cdot m$
            $pro_i = V_i$
            $t_d = t_m, t_u = t_u, t_m = t_d + \lfloor \frac{t_u - t_d}{2} \rfloor$
    DECIDE$(\bigoplus pro_i)$ ▷ The $\bigoplus$ is needed since V_i is a set of sets
function CLASSIFY(V)
    if $|V| \leq t_m$ **then**
        return s
    else
        return m
function FILTER(P)
    $V = \{\forall v \in M | M \in P \wedge M$ rank is greater than $0 \wedge$ ADMISSIBLE$(v, M) \wedge$ M source is in $G \wedge$ v in $E\}$
    return V
function ADMISSIBLE(v, M)
    if $\forall t \in [0, .., |G| - 1]$ such that $G[0:t]$ terminates with s, there exists an admissibility proof for v in M **then**
        return True
    else
        return False
function UPDATEPROOFS(pro_i, P_i, ep)
    for all $v \in pro_i$ **do**
        $proofv = [\perp]^{ep+1}$
        if $ep > 0$ **then**
            for all $t \in [0, \ldots, ep - 1]$ **do**
                if $G[t] = s$ **then**
                    Let p be a valid proof from P_i for $G[0:t]$ (it must exists since v is admissible).
                    $proofv[t] = p$
        Construct a valid proof p for v and group G using messages in P_i.
        $proofv[ep] = p$
        $Proofs = Proofs \cup \{proofv\}$

epoch all values in $E \subseteq \{\{1\}, \ldots, \{n\}\}$ are admissible (i.e., they do not have to carry an admissibility proof), and no classification step is executed. That is, at the end of the epoch each process goes to group $G = s$ and no correct p_i updates its value pro_i. This phase is performed to force Byzantine processes to commit to a certain set of values that cannot be changed later; values that are not associated with a seen-all proof, showing that they have been gradecasted in epoch 0, will be ignored in the

next epochs.

2) Epoch $ep \geq 1$: Epochs 1 and onward share the same structure. At the beginning of an epoch, all correct processes belonging to a same group share the values of three thresholds. At the beginning of epoch $ep = 1$, the group $G = s$ encloses all the processes and the thresholds are $t_d = 0$, $t_m = \frac{n}{2}$ and $t_u = n$.

a) Value gradecast.: An epoch starts by making each correct process p_j in a group G gradecast a message M_j containing, its proposal value pro_i, the group to which p_j belongs, and an *admissibility proof* for each element in pro_i (the structure and the precise purpose of this proof is defined later).

Each other correct process p_i in a group G receives, by mean of the gradecast, a set of tuples: $P_i : \{(p_0, M_0, c_0, S_{p_0,M_0}), (p_1, M_1, c_1, S_{p_1,M_1}), \ldots\}$.

We define a special set of values V_i that is a subset of values in messages contained in P_i. Set V_i contains all "*admissible values*" in P_i and such that: (1) the rank of the message carrying the value is at least 1 and (2) the sender of the message is in G.

Value v is admissible for process p_i in epoch ep if the message that carries v contains also an "*admissibility proof*" proving that v has been seen by all correct processes in $SLV(G[0:1], G[0:2], \ldots, G[0:ep-1])$, where $SLV(Set)$ is a filter function that removes from the set of string Set all the strings that end with letter m. As an example considering $G = ssmsm$ we have $SLV(s, ss, ssm, ssms, ssmsm) = \{s, ss, ssms\}$. Essentially, there must be a proof showing that v has been seen with rank 1 by all processes in the epochs in which processes in G have been classified as slave. The actual structure of this proof is described later (Section IV-B3).

Process p_i becomes a group slave if $|V_i| \leq t_m$, it becomes a group master if $t_m < |V_i|$. If p_i becomes a slave, it enters the group $G \cdot s$. Otherwise, it becomes a master, and it enters $G \cdot m$.

b) Slaves actions: If a process p_i is a slave, it updates its set of thresholds as $t_d = t_d, t_u = t_m, t_m = t_d + \lfloor \frac{t_u - t_d}{2} \rfloor$. Finally, a slave does not update its proposed value pro_i, in the next epoch it will have again the exactly same value it had in the current epoch. Regarding the admissibility proof, a correct slave has the duty to collect an admissibility proof for its value proposed pro_i this is done by collecting the seen-all proof generated by gradecasting its pro_i at the beginning of the epoch.

c) Masters actions: If process p_i is a master, it updates its set of thresholds as $t_d = t_m, t_u = t_u, t_m = t_d + \lfloor \frac{t_u - t_d}{2} \rfloor$. Moreover, it updates its value $pro_i = V_i$.

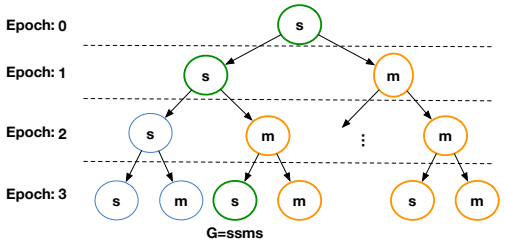

Fig. 3. Graphical representation of the purpose of an admissibility proof.

Regarding the admissibility proof, a correct master has no duty in creating an admissibility proof for its new pro_i, but it has to collect proofs to show that any value inserted in pro_i was admissible in $G[0 : ep-1]$.

3) Admissibility proof: A message m containing a value v carries an admissibility proof for v and group G if message m contains for each, also non-proper, prefix $G[0:j]$ of G terminating with character s (for j in $\{0, 1, \ldots, |G|-1\}$), a seen-all proof for m with a sender p in $G[0 : j-1]$. From Oservation 1, it is immediate to see that an admissibility proof for G implies that all correct processes in $SLV(G[0:1], G[0:2], \ldots, G[0:|G|-1])$ received v in the value gradecast phase and ranked the source of the gradecast at least 1. See Figure 3 for a graphical representation of the usefulness of such a proof. Assume there exists an admissibility proof for value v and group $G = ssms$, then there is a seen-all proof for the epochs $0, 1, 3$ and groups in $G_s : \{s, ss, ssms\}$(marked as green in the figure). This implies that, in each of these epochs, value v has been seen by each correct process with rank at least 1. In particular, a seen-all proof for value v and group ss implies that v has been gradecasted by a process in group s, and it has been received by all correct with rank at least 1. This implies that v has been inserted in the proposal of all correct masters in group sm (in the figure we represent with an orange border the processes that have v in their proposal). This means that a master in ssm can update its proposal inserting v, and it knows that it will still be comparable with the decision of any process in a group with prefix sm. Iterating the reasoning, a chain of seen-all proofs, the first for group ss and the second for group $ssms$, implies that v is in the proposal of all correct processes in a group with prefix sm or $ssmm$, and thus a future master in $ssmmsm$ can safely include v in its proposal. The necessity of a seen-all proof for epoch 0, and thus for group s is needed to force Byzantine processes to commit to at most f values. This is due to the fact that f Byzantine processes are able to create admissibility proofs for at most f distinct

values in epoch 0.

4) Termination: A process p_i terminates the above algorithm when the epoch is $\log(n)+1$. Upon termination it decides its value pro_i.

C. Correctness of GAC

Definition 1. *Let $A(G)$ be the set of values admissible for correct processes in group G during the gradecast of epoch $|G|$.*

Given a group G the lemma below shows that the set of admissible values of any other group $G' = G \cdot \sigma$ will be a subset, not necessarily proper, of $A(G)$.

Lemma 2. *Let us consider the group G it holds $A(G') \subseteq A(G)$ for any $G' = G \cdot y$.*

Proof. The proof is by induction on y:

- **Base case.** $G' = G \cdot \epsilon$: It is immediate by observing that $G' = G$. Thus $A(G) = A(G')$.
- **Inductive case.** We assume the above is true up to $G' = \sigma$, in the inductive step we have to show that it holds for the two possible extensions of σ. Case (1): $G' = \sigma \cdot m$, in such a case the set of admissible values does not change. Thus $A(\sigma) = A(G')$. Case (2): $G' = \sigma \cdot s$, suppose that a value v is in $A(G')$ but not in $A(\sigma)$. In order to be admissible for G' there must exist a proof for each prefix of G' ending with an s, this is by construction also an admissibility proof for σ. Therefore $A(G') \subseteq A(\sigma)$. □

Definition 2. *Given an epoch ep we define as W_{ep} the set of values that have been gradecasted by a process in epoch ep and that have been seen by all correct processes with gradecast rank at least 1. With $W_{ep}(G)$ we indicate the subset of W_{ep} that was sent by processes claiming to belong to group G.*

Lemma 3. *Consider a correct master p_i in the group $G = \sigma \cdot m$. If p_i decides, its decision, let it be dec_i, is comparable with the one of any correct slave in $G' = \sigma \cdot s$.*

Proof. To prove the above it is sufficient to show that each value v in $A(G')$ is included in dec_i. In order for a value to be in $A(G')$ it must exist an admissibility proof for group G', the admissibility proof implies two things: (1) that v is in $W_{|\sigma|}(\sigma)$ (see Observation 1); (2) that v is in $A(\sigma)$. From the above, and the code of a correct master p_i, v will be in the set V_i of Line 11 at epoch $|\sigma|$. Since a master never removes a value from its pro_i its decision dec_i must include v. This complete the proof since each correct slave p in G' never put in its proposal a value that is not in $A(G')$: by Lemma 2 the set of values that a slave will consider in any possible future execution of Line 11 is included in $A(G')$. □

Lemma 4. *Let p be a correct process that decides dec_i. Its decision respects Inclusivity and Non-Triviality.*

Lemma 5. *Let p be a correct process that at some epoch $ep \leq \log(n)+1$ updates its proposal pro_i with a new value w. For any possible G such that $p \in G$ and $\log(n)+1 \geq |G| \geq ep$ we have $w \in A(G)$.*

Lemma 6. *For any correct process $p_i \in G$ with $G \neq \epsilon$ we have $|A(G)| - |pro_i| \leq n/2^{|G|-1}$. Moreover, given process $p_i \in G$ we have $|A(G)| \leq t_u$ and $pro_i \geq t_d$, where t_u and t_d are the thresholds of group G.*

Proof. We will show that for each $p_i \in G$ it holds that $|A(G)| \leq t_u$, that $|pro_i| \geq t_d$, and, that $t_u - t_d \leq n/2^{|G|-1}$. Recall that t_u, t_d and t_m depend on G. The proof is by induction on G.

- Base case: $G = s$: it derives immediately from the structure of the lattice and the fact that $t_d = 0$ and $t_u = n$ for all processes in G.
- Inductive case: By inductive hypothesis the claim holds for σ. We have $G = \sigma \cdot m$ or $G = \sigma \cdot s$. Let t_u, t_m, t_d be the thresholds of group σ, by inductive hypothesis we have $|A(\sigma)| \leq t_u$ that $|pro_i| \geq t_d$ and that $t_u - t_d \leq n/2^{|G|-1}$.
  - Case $G = \sigma \cdot m$. First notice that for master processes the set of admissible values does not change, that is $A(\sigma) = A(G)$ neither the threshold t_u. In such a case we will show that the lattice "shrinks from below" in the sense that by updating the lower bound on pro_i our claim holds. Since p_i is in G we have that it updates pro_i and the new pro_i contains a number of elements that are at least $t_d + \lfloor \frac{t_u - t_d}{2} \rfloor + 1$, thus the new $t'_d = t_d + \lfloor \frac{t_u - t_d}{2} \rfloor + 1$. By immediate algebraic manipulations we have $t_u - t'_d \leq \frac{t_u - t_d}{2}$ that proves our claim.
  - Case $G = \sigma \cdot s$. In such a case we will show that the lattice "shrinks from above". Consider the generic $p_i \in G$ this implies that at the end of epoch $|\sigma|$ the set V_i contained at most t_m elements. It is immediate that since p_i is correct each value in $A(\sigma)$ that is not in V_i cannot be admissible in the extension G. Therefore we have $|A(G)| \leq t_m$, now it remains to show that $|pro_i| \geq t_d$ but this is immediate from inductive hypothesis. Thus we have $t'_u = t_m$ and $t'_d = t_d$, that implies $t'_u - t'_d \leq t_m - t_d \leq \frac{t_u - t_d}{2}$. □

Observation 2. *Any correct process decides at epoch $\log(n)+1$.*

Lemma 7. *Given any pair p_i, p_j of processes in C their decision dec_i and dec_j are comparable.*

Proof. Let G_i be the group where p_i belongs at the end of epoch $r = \log(n) + 1$, and let G_j the analogous for p_j. If G_i and G_j share a common prefix σ such that $\sigma \cdot m$ is a prefix of G_i and $\sigma \cdot s$ is a prefix of G_j, then Lemma 3 shows the comparability.
The only case when the above (or the symmetric of the above) does not hold is if $G = G_i = G_j$. In this case, we will show that $dec_i = dec_j$. Suppose the contrary, then we have that there exists at least a value $v \in dec_i$ and such that $v \notin dec_j$. By Lemma 5 we have $v \in A(G)$, and by Lemma 2 v is in each prefixes of G. Suppose p_i inserted v in its proposal in an epoch ep such that it has been master again in epoch $ep' > ep$, however this implies that also p_j is master in ep', and p_i being correct in epoch ep' it gradecasts v that will be included in the proposal of p_j. The above implies that v has been received and inserted by p_i in its proposal exactly in the last epoch in which p_i became master.

Let ep_{last} be such an epoch. Note that if $t_u - t_d \leq 2$ then v is also in the proposal of p_j (both processes enter in the master group in ep_{last}): in case $t_u - t_d = 2$ to become master each process has to collect enough values to trespass the threshold t_m, recall that $t_m = t_d + 1$ and thus it has to collect t_u values, but those and are all the admissible values (by Lemma 6). In the other case, when $t_u - t_d = 1$, a process has also to collect all admissible values ($t_m = t_d$ and $t_u = t_m + 1$) (by Lemma 6).

Therefore, $t_u - t_d > 2$ in ep_{last}, however by the structure of the algorithm and the number of epochs being $\log(n) + 1$ we eventually have an epoch $ep' > ep_{last}$ such that $t_u - t_d = 1$ and $t_m = t_d$, when this happens process p_j will become master upon receipt of v from p_i (by Lemma 5 v is admissible for p_j). This contradicts the fact that $G_i = G_j$, since p_i is never again a master after epoch ep_{last}. □

From previous lemmas we have:

Theorem 2. *Given the lattice L constituted by the power set of the first n natural numbers with union as join operation, GAC is a correct LA algorithm on L, that terminates in $\mathcal{O}(\log(n))$ rounds and tolerates up to $n/3 - 1$ Byzantine processes.*

V. Adapting GAC to work on arbitrary semi-lattices in $\log(f)$ rounds

We first explain how to adapt the GAC algorithm to work in $\log(f)$ on L_n when each correct proposes a different unique value in $\{\{1\}, \{2\}, \ldots, \{n\}\}$. We call such an algorithm GAC$_{fast}$, we then discuss how to adapt GAC$_{fast}$ to work on a generic join semi-lattice dropping the assumption of different proposal values. The main idea is to modify epoch 0 to satisfy two needs: (1) to force Byzantines to commit to a certain value; (2) to make all processes collect at least $n - f$ different proposal values. This allows the thresholds to be set to $t_d = n - f, t_u = n, t_m = \lfloor n - \frac{f}{2} \rfloor$ in all processes at the end of epoch 0. The modified epoch 0 is Algorithm 3. The code follows the old one with the notable exceptions that: a correct process updates its proposal by including all values, in E, that have been seen with rank at least 2, and it updates its thresholds accordingly.

Algorithm 3 GAC$_{fast}$: Collect and Commit Epoch 0 - Algorithm for process p_i

```
1: G = ϵ, pro_i, Proofs = {}
2: function LA-PROPOSE(pro_i)
3:     pro_i = pro_i
4:     GRADECAST(M = (pro_i, G, ⊥))                ▷ Epoch 0 - start
5:     P_i = RCV()
6:     V_i = {∀v ∈ M | M ∈ P_i ∧ M rank is equal 2 ∧ v ∈ E}
7:     G = G · s
8:     pro_i = V_i
9:     updateproofs(pro_i, P_i, 0)                 ▷ Epoch 0 - end
10:    t_d = n − f, t_u = n, t_m = t_d + ⌊(t_u − t_d)/2⌋
11:    ...                  ▷ Remain as Algorithm 2 but for line 8
```

The remaining of the algorithm is the same as Algorithm 2 but for line 8 where we have $ep \in [1 \ldots, \log(f) + 1]$.

a) Correctness discussion: The same lemmas and observations of Section IV-C hold with the following exceptions:

Lemma 8. *For any correct process $p_i \in G$ with $G \neq \epsilon$ we have $|A(G)| - |pro_i| \leq f/2^{|G|-1}$. Moreover, given process $p_i \in G$ we have $|A(G)| \leq t_u$ and $pro_i \geq t_d$, where t_u and t_d are the thresholds of group G.*

Theorem 3. *Given the lattice L constituted by the power set of the first n natural numbers with union as join operation and where each correct process proposes a distinct element in $\{\{1\}, \ldots, \{n\}\}$, GAC$_{fast}$ is a correct LA that terminates in $\mathcal{O}(\log(f))$ rounds and tolerates up to $n/3 - 1$ Byzantine processes.*

A. Arbitrary Semi-lattices L_A

We adapt GAC$_{fast}$ to work on an arbitrary join semi-lattice $L_A = (V_A, \oplus)$, an arbitrary set $E_A \subseteq V_A$ of allowed proposal values, and an arbitrary mapping of proposal values and correct processes (recall that in previous section we were assuming a different proposal value for each correct). The adaptation works by running

GAC$_{fast}$ on an intermediate semi-lattice L^*. Lattice L^* is the one induced by the union operation over the power set of $V^* = \Pi \times E_A$. The set V^* is constituted by all possible pairs process ID and initial proposed value pro_i (each pro_i is in E_A). Each correct process p_i starts GAC$_{fast}$ with input (p_i, pro_i).

Note that epoch 0, with is commitment functionality, forces the algorithm to effectively decides on a lattice L^* that is the power set of a subset X of V^* of cardinality at most n and at least $n-f$. Such a lattice is isomorphic to the lattice on which the correctness of GAC$_{fast}$ has been shown in Section V. Once GAC$_{fast}$ terminates each process $p_i \in C$ has a decision dec_i. This decision dec_i is a set $\{(p_i, val_i), \ldots, (p_x, val_x)\}$, from such a set the process p_i obtains a decision dec'_i on L_A where dec'_i is $dec'_i = \bigoplus D_i$ given $D_i : \{y | \forall (x,y) \in dec_i \wedge y \in E_A\}$. This strategy enforces that the decisions dec'_i and dec'_j of any two correct processes p_i, p_j are comparable points on the semi-lattice L_A. It is also immediate that non-triviality and inclusivity hold. From the above we have:

Theorem 4. *Given f Byzantine processes and n processes in total, if $n \geq 3f + 1$, there exists a Byzantine lattice agreement algorithm terminating in $\mathcal{O}(\log f)$ rounds in the authenticated message model.*

1) Message Complexity: The provable gradecast generates at most $\mathcal{O}(n^2)$ messages at each round. Each epoch is composed by 3 rounds and in each epoch all correct processes do a gradecast, thus we have a total of $\mathcal{O}(n^3 \log f)$ messages. However, as pointed out in [6], it is possible to use $\mathcal{O}(n^2)$ messages in total to run n parallel instances of gradecast (each message will be structured with n locations one for each possible gradecaster). Therefore, our algorithm can be implemented using $\mathcal{O}(n^2 \log f)$ messages.

VI. Trade-off Between Signatures and Number of Processes

Signatures are used to implement the seen-all proof of our provable gradecast primitive (explained in Section IV-A). By assuming $n \geq 4f + 1$ processes we may implement an interactive version of the seen-all proof that does not use signatures. We explain the interactive provable gradecast algorithm when $n = 4f + 1$, the extension for $n > 4f+1$ is immediate. The modifications with respect to Section IV-A are as follows:

- The threshold of line 23 remains $f + 1$, the one of line 20 becomes $3f + 1$, the threshold of line 14 is $3f + 1$. Therefore, a message will have rank 1 if seen with multiplicity at least $f + 1$, a message has rank 2 if seen with multiplicity at least $3f + 1$.
- The seen-all proof S_{p_s,m_i} is simply a set of IDs. These IDs are the ones of processes from which p_i receives m_i in the receive phase of round 3.

The seen-all proof S_{p_s,m_i} is checked in an interactive way by querying each process contained in the set S_{p_s,m_i}. The proof passes if at least $2f + 1$ of such processes confirm to have relayed m_i in the relevant gradecast instance. It is obvious that by increasing the two thresholds we do not affect the original properties of the gradecast (discussed in Section IV-A).

Using the interactive provable gradecast and the straightforward interactive variant of the admissibility proof we have:

Theorem 5. *Given f Byzantine processes and n processes in total, if $n \geq 4f + 1$, then there exists a byzantine lattice agreement algorithm terminating in $\mathcal{O}(\log f)$ rounds in the authenticated channel model.*

We argue that the interactive provable gradecast generates at most $\mathcal{O}(n^2)$ messages at each round. The additional cost introduced by the interactive proof is at most $\mathcal{O}(n)$ per round. Thus the total asymptotic cost remains the same: $\mathcal{O}(n^2 \log f)$ messages.

VII. An universal transformer from LA to Generalised LA

In this section we show a transformer algorithm that builds upon a LA algorithm to create a Byzantine tolerant Generalised LA algorithm. We consider the definition of [7] adapted for a synchronous system. In the Synchronous Generalised LA, each correct process p_i receives input values from an infinite sequence $Pro_i = \langle pro_0, pro_1, pro_2, \ldots \rangle$ where each pro_k is a value inside a set of admissible values E (note that E is not necessarily finite). Without loss of generality we imagine that at each round r, p_i receives a value $pro_r \in Pro_i$ (note that this is not restrictive since we could modify the lattice to admit a neutral element, such as $\emptyset$). A correct process p_i must output an infinite number of decision values $Dec_i = \langle dec_0, dec_1, dec_2, \ldots \rangle$. The sequence of decisions has to satisfy the following properties:

- **Liveness:** each correct process $p_i \in C$ performs an infinite sequence of decisions $Dec_i = \langle dec_0, dec_1, dec_2, \ldots \rangle$;
- **Local Stability:** For each $p_i \in C$ its sequence of decisions is non decreasing (i.e., $dec_h \subseteq dec_{h+1}$, for any $dec_h \in Dec_i$);
- **Comparability:** Any two decisions of correct processes are comparable, even when they happen on different processes;

- **Inclusivity:** Given any correct process $p_i \in C$, if Pro_i contains a value pro_k, then pro_k is eventually included in $dec_h \in Dec_i$;
- **Non-Triviality:** Given any correct process $p_i \in C$ if p_i outputs some decision dec_k at a round r, then $dec_k \leq \bigoplus(Prop[0:r] \cup B[0:g(r)])$, where, $Prop[0:r]$ is the union of the prefixes, until index r, of all sequences Pro_i of correct processes; and, $B[0:g(r)]$ is the union of all prefixes, until index r, of f infinite sequences B_i, one for each Byzantine process. Function g is $g : \mathbb{N} \to \mathbb{N}$. Each B_i is a sequence of elements in E.

Intuitively, function g upper bounds the number of values that Byzantine processes can insert.

a) Transformer: We now explain the high level idea behind the transformer. Let $\mathcal{LA}$ be a one shot synchronous lattice agreement algorithm that terminates in δ rounds. We divide the time in terms, a term lasts for δ rounds and it allows to execute, from start to termination, an instance of $\mathcal{LA}$. At the beginning of term k, correct processes start the k-th instance of $\mathcal{LA}$, we denote it as k-$\mathcal{LA}$. Each correct process receives from upper layer a stream of elements in E, and it batches such elements until a new instance of $\mathcal{LA}$ starts. Let C_k be the k-th batch, at the beginning of term k, process p_i starts the instance k-$\mathcal{LA}$ with input $(p_i, dec_{k-1} \oplus C_k)$, where dec_{k-1} is the output of the $(k-1)$-$\mathcal{LA}$ instance. There are few minor details to add to this description to get an actual algorithm (see pseudocode in Algorithm 4), the most important being the mechanism needed to bound the number of values that could be added by Byzantine processes. The key idea is to start the instance k-$\mathcal{LA}$ with a set of admissible values $P_{T(k-1)+\delta}$, that is the set of all subsets of E of size less or equal to $T(k-1)+\delta$. Function T is a function mapping each index of the decision sequence of a correct process to an upper bound on the maximum size of the decision, where the size is counted as number of elements in the decision. We assume $T(-1) = 0$; we have that $T(0) = \delta \cdot n$: each correct process that starts the first instance of LA proposes at most δ values; the closed form for $k > 0$ of $T(k)$ is in the statement of Lemma 9.

b) Correctness discussion: Assuming that each $\mathcal{LA}$ is an instance of a correct LA algorithm (according to definition in Section II-0a), we argue that the Generalised LA algorithm obtained by using our transformer is a correct algorithm for definition in Section VII. The liveness property is satisfied by the liveness of each instance of LA. The local stability and the inclusivity derive directly from the fact that once a process p outputs dec_{k-1}, the next instance of LA will have as input a value that contains $(p, dec_{k-1} \oplus C_k)$. Therefore, by inclusivity of LA we have that the decision of k-$\mathcal{LA}$ contains the pair $(p, dec_{k-1} \oplus C_k)$, and this means that $dec_{k-1} \oplus C_k \leq dec_k$. It remains to show the non-triviality:

Lemma 9. *Consider the sequence of decisions of a correct process p executing the transformer in Algorithm 4. Each dec_k in the sequence, decided at round r, respects the Non-Triviality property for a function $g(r) < (T(k) = \frac{\delta \cdot n((f+1)^{k+1}-1)}{f})$.*

Algorithm 4 *Transformer* - Algorithm for process p_i

1: $C = \emptyset, dec = \emptyset$ ▷ Batch of values, Decision
2: **upon event** PROPOSE(pro_r)
3: $C = C \cup \{pro_r\}$
4: **upon event** $r = \delta \cdot (k+1)$ FOR SOME $k \in \mathbb{N}$
5: Start the instance k of LA over lattice $\mathcal{L}_k$ with admissible values $E_k = \Pi \times P_{T(k-1)+\delta}$
6: k-LA-PROPOSE$((p_i, dec \oplus C))$
7: $C = \emptyset$
8: **upon event** DECISION FROM THE CURRENT INSTANCE OF LA(dec')
9: $X : \{y | (x, y) \in dec'\}$
10: $dec = \bigoplus X$
11: Decision$_k(dec)$

REFERENCES

[1] H. Attiya, M. Herlihy, and O. Rachman, "Atomic snapshots using lattice agreement," *Dist. Comp.*, vol. 8, no. 3, 1995.

[2] M. Raynal, *Fault-tolerant Agreement in Synchronous Message-passing Systems*. Morgan & Claypool, 2010.

[3] G. A. Di Luna, P. Flocchini, T. Izumi, T. Izumi, N. Santoro, and G. Viglietta, "Population protocols with faulty interactions: The impact of a leader," *Theor. Comput. Sci.*, vol. 754, pp. 35–49, 2019.

[4] G. A. Di Luna, P. Flocchini, G. Prencipe, N. Santoro, and G. Viglietta, "Line recovery by programmable particles," in *ICDCN 2018*, 2018, pp. 4:1–4:10.

[5] O. Michail, P. G. Spirakis, and M. Theofilatos, "Fault tolerant network constructors," *CoRR*, vol. abs/1903.05992, 2019. [Online]. Available: http://arxiv.org/abs/1903.05992

[6] X. Zheng and V. K. Garg, "Byzantine lattice agreement in synchronous systems," *CoRR*, vol. abs/1910.14141, 2019, http://arxiv.org/abs/1910.14141.

[7] G. A. Di Luna, E. Anceaume, and L. Querzoni, "Byzantine generalized lattice agreement," in *(to appear) IPDPS 2020*, 2020, https://arxiv.org/abs/1910.05768.

[8] J. M. Faleiro, S. Rajamani, K. Rajan, G. Ramalingam, and K. Vaswani, "Generalized lattice agreement," in *PODC*, 2012.

[9] H. Attiya and J. Welch, *Distributed Computing*. John Wiley & Sons, Ltd, 2004.

[10] X. Zheng, C. Hu, and V. K. Garg, "Lattice agreement in message passing systems," in *DISC 2018*, 2018.

[11] T. Nowak and J. Rybicki, "Byzantine approximate agreement on graphs," in *DISC 2019*, Dagstuhl, Germany, 2019, pp. 29:1–29:17.

[12] G. Di Luna, E. Anceaume, S. Bonomi, and L. Querzoni, "Synchronous byzantine lattice agreement in log f rounds," https://arxiv.org/abs/2001.02670, 2020, arXiv:2001.02670v2.

[13] M. Ben-Or, D. Dolev, and E. Hoch, "Simple gradecast based algorithm," arXiv preprint, Tech. Rep. arXiv:1007.1049, 2010.