REVISITING BYZANTINE AGREEMENT AND MPC IN BOUNDED-DELAY NETWORKS

A Thesis

by

FATMAELZAHRAA ELSHEIMY

Submitted to the Graduate and Professional School of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee, Juan Garay

Committee Members, Yupeng Zhang

Jeyavijayan Rajendran

Head of Department, Riccardo Bettati

August 2022

Major Subject: Computer Science

ABSTRACT

Secure multi-party computation (MPC) is one of the most important and fundamental problems in distributed computing. MPC allows a set of parties to compute an arbitrary function of their (potentially private) inputs in a secure way even in the presence of an adversary. The problem has been studied extensively in the synchronous setting, where it is assumed that there exists a global clock and the delay of any message in the network is bounded. However, though theoretically impressive, such networks do not model adequately real-life networks, like the Internet.

In this work, we present the first construction for MPC in the partial synchronous setting, where there's an unknown bounded delay on the message delivery time. Our protocol achieves optimal resilience, involving $n = 3t + 1$ parties and tolerating a malicious adversary, capable of corrupting up to t parties.

DEDICATION

This thesis was done under the supervision of Juan Garay. I thank him for introducing me to a fascinating field of research, for his competent guidance, as well as for many inspiring discussions. On top of being an expert on experts, he is a devoted advisor.

I would like to thank Yupeng Zhang for his countless support on both personal and professional levels. I have enjoyed, and learned a lot from interacting with him. I dearly cherish our friendship. I thank Jeyavijayan Rajendran for his time and for spontaneously accepting to be a committee member of this thesis.

I would like to thank my family for the endless support and love. I wouldn't have finished this work without their help. A special feeling of gratitude to my loving mom, Najlaa. The sacrifices she made, love she shared, and support she gave me is the only reason I've become the person I am today.. She is all my reasons.

I would like to thank my friends Nouran, Christian, Daniel, Sarah, and Sanaa who have supported me throughout the process. You have been my best cheerleaders.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a thesis committee consisting of Professor Juan Garay [advisor] and Yupeng Zhang of the Department of [Computer Science] and Professor Jeyavijayan Rajendran of the Department of [Elecctrical Engineering].

All the work conducted for the thesis was completed by the student independently.

Funding Sources

Graduate study was supported by a Teaching Assistantship from Texas A&M University.

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

1. INTRODUCTION AND LITERATURE REVIEW

Consensus algorithm (Byzantine Agreement), introduced by Lamport et al., [LSP82], is a fundamental primitive in distributed computing. The traditional setting in which the problem was introduced and investigated considers synchronous communication and protocol execution, where no network delay is assumed. Since studying the Byzantine problem in synchronous setting is unrealistic when relating it to distributed computing, Dwork et al, [DLS88] extended the analysis and studied the problem in the partial synchronous setting. According to [DLS88], in a partially synchronous system: there's a fixed unknown upper bound on the delay that the adversary may inflict to the delivery of any message; and there's a fixed unknown upper bound on the relative speed of different processors. Dwork et al, proved that with or without public key infrastructure(PKI), the protocol can't tolerate more than n/3 malicious parties.

With the introduction of "Nakamoto" Consensus and blockchains e.g., Bitcoin [Nak09], it demonstrated how can Consensus and Broadcast can be reached even in settings where a majority of the participants might be adversarial, as long as the majority of the computation power remains honest. Similar approaches were also used for alternative blockchains that relied on other different assumptions about restricting different resources, such as proof of stake, proof of space,...etc. In "Nakamoto" consensus [Nak09], parties engage and organize transactions in a ledger without having any information about each other (Public Key Infrastructure) or knowledge of the number of parties participating in the protocol at any given time. This contrasts with classical models and results in consensus (Byzantine agreement).

This proposes the following question:

Could the resource-restricted paradigm be generically used to circumvent the classical strong impossibility results or is the mismatch due to different model and goals?

This work extends on the the initial formulation of RRC [Gar+20], where it considers cryptographic protocols executing in an idealized synchronous network. Specifically, we address the open applicability and relevance of the framework to more realistic network models that take into account both communication delays and parties operating at different speeds due to different clock rates—the so-called "partial synchrony" model (cf. [**dls**], [Bad+19]]).

This work also studies various cryptographic primitives in the bounded-delay network. It has come to our attention how understudied this field is and we wanted to fill in the gap. As mentioned earlier, the bounded-delay network is much more practical and relevant than the intensively-studied synchronous setting. We believe that this work will be a milestone for the current-emerging applications, which their security is depended on the bounded-delay network.

2. The justification and Significance of the Study

Byzantine agreement (BA), introduced by Lamport, Shostak, and Pease, is a well studied problem, and and is at the core of many secure multi-party computation (MPC) protocols.[LSP82]. The problem comes in two flavors; Consensus and Broadcast. Consensus considers a set of n parties P = $\{P_1, \ldots, P_n\}$ each of whom has an input x_i, and who wish to agree on an output y (Consistency) such that if x_i = x for all honest parties then y = x (Validity), despite the potentially malicious behavior of up to t of them. In the Broadcast version, on the other hand, only a single party, often called the sender has an input x_s, and the goal is to agree on an output y (Consistency) which, when the sender is honest equals x (Validity).

The problem was originally studied in the synchronous setting, where parties proceed in rounds at the same time and each party has a consistent view of the current round. The underlying communication network is a complete point-to-point authenticated channels network, where every pair (P_i, P_j) of parties is connected by a channel, such that when P_j receives a message on this channel it knows it was indeed sent by P_i. In this synchronous setting, [LSP82] proved that there exists no Consensus or Broadcast protocol which can tolerate $t \geq n/3$ Byzantine parties, i.e., parties controlled by a (central) active and malicious adversary. This is in the information theoretic model (security with zero error probability) and no correlated randomness shared among the parties.

However, using PKI(public-key infrastructure), Dolev and Strong [DS83] proved that assuming a PKI implying existentially unforgeable digital signatures (e.g., one way functions) Broadcast tolerating arbitrarily many (i.e., $t < n$) malicious corruptions is possible.

In world-scale systems, it is difficult to argue for synchronous communication. Consequentially, the focus has shifted away from synchronous systems towards asynchronous systems. In the asynchronous setting model, for any message sent, the adversary can delay its delivery by any finite amount of time. So, on the one hand, there is no bound on the time to deliver a message but, on the

other hand, each message must eventually be delivered. It is much more challenging for an honest party to assess whether another party is malicious, if that party's messages can be indeterminately delayed, but this scenario much better reflects that network reliability in the real world.

Unfortunately, the fundamental impossibility result of [FLP85] demonstrates that there is no deterministic algorithm for achieving agreement in the asynchronous setting even against benign failures. One solution which overcomes this problem, first introduced by Rabin [Rab83] and Ben-Or [Ben83], is to use randomisation.

To circumvent the [FLP85] impossibility result and to have a more flexible model, the notion of partial synchrony was introduced by Dwork, Lynch and Stockmeyer in [DLS88]. A partial synchrony model captures the intuition that systems can behave asynchronously (i.e., with variable/unknown processing/communication delays) for some time interval, but that they eventually stabilize and start to behave (more) synchronously. Therefore, the idea is to let the system be mostly asynchronous but to make assumptions about timing properties that are eventually satisfied.

Dwork et al,. [DLS88] introduced to types of partial synchronous model;

Model 1:

1. there's a fixed unknown upper bound $\triangle$ on the delay (measured in number of rounds) that the adversary may inflict to the delivery of any message.

2. there's a fixed unknown upper-bound ϕ on the drift between any two parties

Model 2: For each execution, there is an unknown global stabilization time GST,where after GST

1. there's a fixed known upper bound $\triangle$ on the delay (measured in number of rounds) that the adversary may inflict to the delivery of any message.

2. there's a fixed upper-bound ϕ on the drift between any two parties

[DLS88] determines the maximum resiliency possible for fault models in partial synchronous setting. The following table summaries the paper's results.

The blockchain era With the recent introduction of blockchain, "Nakamoto" consensus violated the information-theoretic bounds{ no consensus for $t \geq n/3$}, showing that even a majority of corrupted parties can be tolerated as long as the majority of the computation resources remain at honest hands, even without public key infrastructure.

An intriguing question comes to mind, can the dolev and strong impossibility bounds, where they consider [LSP82] model (of instant delivery authenticated channels and full synchrony) be circumvented under the resource-restricting paradigm?

[Gar+20] answered this question in the affirmative, by representing resource-restricting paradigm as an access restriction on the underlying communication network. Basically, the parties are restricted from sending unbounded messages to other parties. The paper constructs the model in Canetti's Universal Composition framework [CA02], representing the restriction of the resources available to the adversary by means of a functionality wrapper, which wraps a communication network and restricts the ability of parties (or the adversary) to send new messages through this network. The paper only deals with synchronous setting.

In this work, we build on the [Gar+20] paper, by extending the work to the partial synchronous setting. We attempt to circumvent the bounds in [**dls**] using the same idea of the resource-restriction paradigm. We attempt to answer whether the same model could be used to reach a better resiliency in the partial synchronous bound or not.

3.0.1 Cryptographic Primitives

Protocols (or algorithms) take an input x, from a certain domain D, perform a computation x and gives an output y = f(x). There are two types of algorithms; deterministic and randomized. Deterministic means that given a certain input, it will always output the same value y. Unlike deterministic, randomized algorithms' output depend on the randomness factor. Meaning, it doesn't always output the same thing. Essential properties of algorithms are whether they output the correct computation, and how long they take to halt.

In a deterministic algorithm, if it has a polynomial upper-bounded computational worst-case complexity, then we say it is called polynomial. For probabilistic algorithms, there are two types of algorithms; Monte-Carlo and Las Vegas. For Monte-Carlo, all inputs produce the correct output with a probability greater than 1/2, however, the protocol always terminates. For Las vegas, the protocol isn't guaranteed to always halt, however, whenever it does, it always outputs the correct output. The protcol terminates with probability greater than 1/2 as well.

The main goal for cryptographic primitives is to provide secrecy and authenticity of information. We say a protocol has negligible probability if it is exponentially small in the parameter k. To prove a protocol is secure, we have to show that the adversary can't break it with a probability greater than negligible.

3.0.2 Cryptographic Protocols

Protocol could be stated informally as a game between some players. Each player is able to send and receive messages, perform a computation, and could also have access to a random oracle. When we are talking formally, we say each player is a turing machine. Each turing machine has an input tape, computation tape, and an output tape. If the turing machine is probablistic, then it has a random tape as well.

The input tape is where the machine receives its communication from other parties, and it is a read-only tape. The computation tape is a read and write tape. While, the output tape is a write-only tape. Furthermore, all turing machines have pairwise communication; they all share tapes with one another. If a turning machine wants to send data to another turing machine, it will right on the other's machine input tape.

When we say a machine took execution step, it means it received a finite number of input messages from other parties, did some finite computation and made an output to another turing machine.

A protocol is either deterministic or randomized. Deterministic means that given a certain input, it will always output the same value y. Unlike deterministic, randomized algorithms' output depend on the randomness factor. We say a protocol terminates if all party terminates.

3.0.3 Communication

For all algorithms, we assume a fully connected point-to-point network. This means to all players share read-only and write-only tapes with each other. If we are talking about information-theoretic protocols, then all these channels are secure. This means that an unconditionally-bounded adversary can't see any messages between honest parties. Channels are called "authenticated channels", if the parties know who they received the message from and who they are sending the message to. Channels are called "private", if the adversary can't see the messages between honest parties. Channels are "secure" if they are both private and authenticated.

A protocol could have a broadcast primitive, meaning that each party could send one message to everyone, and all parties will receive the same exact message. Among n parties, there are n possible way to construct broadcast channels, each party could act as a sender. A broadcast primitive ensures that all parties receive the same message, even if the sender acted maliciously.

3.0.4 Synchronicity

Synchronicity setting is a major factor when studying the protocol. Network could either synchronous, partial synchronous or asynchronous.
Synchronous:

- There's a fixed known upper bound Δ on the delay (measured in number of rounds) that the adversary may inflict to the delivery of any message.
- There's a fixed upper-bound α on the drift between any two parties

Partial Synchronous:

- there's a fixed known upper bound $\triangle$ on the delay (measured in number of rounds) that the adversary may inflict to the delivery of any message.
- there's a fixed upper-bound ϕ on the drift between any two parties

Asynchronous:

- No upper-bound on the message delay
- Eventual-delivery

3.0.5 Adversary

There are two types of adversaries; malicious and passive. A passive adversary follows the protocol, but try to gain information that's not authorized to. A malicious adversary may steer away from the protocol. He could either halt at a random point of the protocol(fail or stop), send wrong

messages or do wrong computation (malicious). A malicious adversary can read all the infomration of its corrupted party and We say a protocol is t-resilient if it could withstand up to t-adversaries. Usually, t is a function of the n number of players. We say a protocol assumes honest majority if it requires $t < n/2$

3.0.6 Computational Power

The adversary could either has unconditional or conditional power. Conditional power if it is in cryptographic setting. Meaning that the setting depends on some hard mathemtical assumptions such as deffie helman or factoring of primes problem. An information-theoretic setting is when the adversary has unconditional-power. Note that in the information-theoretic setting, we assume that the channels between honest parties are secure

4. Byzantine Generals Problem

Byzantine agreement (BA), introduced by Lamport, Shostak, and Pease, is a well studied problem, and and is at the core of many secure multi-party computation (MPC) protocols.[LSP82]. The problem comes in two flavors; Consensus and Broadcast. Consensus considers a set of n parties P = $\{P_1, \ . \ . \ . \ , P_n\}$ each of whom has an input x_i, and who wish to agree on an output y (Consistency) such that if x_i = x for all honest parties then y = x (Validity), despite the potentially malicious behavior of up to t of them. In the Broadcast version, on the other hand, only a single party, often called the sender has an input x_s, and the goal is to agree on an output y (Consistency) which, when the sender is honest equals x (Validity).

The problem was originally studied in the synchronous setting, where parties proceed in rounds at the same time and each party has a consistent view of the current round. The underlying communication network is a complete point-to-point authenticated channels network, where every pair (P_i, P_j) of parties is connected by a channel, such that when P_j receives a message on this channel it knows it was indeed sent by P_i. In this synchronous setting, [LSP82] proved that there exists no Consensus or Broadcast protocol which can tolerate $t \geq n/3$ Byzantine parties, i.e., parties controlled by a (central) active and malicious adversary. This is in the information theoretic model (security with zero error probability) and no correlated randomness shared among the parties.

5. Multi-Party Computation

Multi-party computation is one of the most fundamental problems in distributed computing. First we provide an ideal definition for MPC. A computation in this ideal model proceeds as follows. Beginning for a model to be ideal and has an ideal-model adversary. The adversary would chooses to corrupt some parties as it desires. The corruption could be malicious or passive. The adevrsray first learns their inputs, and perhaps changes it. After that, all parties give their inputs to a trusted entity. The trusted party then computes the output based on the inputs given to it. The trusted party then hands it to all parties. We don't care about what the corrupted parties' output. The honest parties output the value received from the trusted party. We then say an MPC protocol is secure if the execution in the real world is equivalent to the execution in the ideal model.

MPC must maintain the following properties:

- Correctness: The output must be the correct value of running the function on the parties' inputs
- Privacy: none of the parties' inputs is leaked.

MPC problem was intensively studied in different setting; synchronous, partial synchonous and asynchronous. In the information-theoretic setting: the protocol can withstand up to $t < n/3$ malicious parties in the synchronous setting, and $t < n/4$ in the asynchronous setting. In the cryptographic setting, a protocl could withstand up to $t < n/2$ in the synchronous setting, and $t < n/3$ in the asynchronous setting.

6. Resource-Restricted Cryptography

The blockchain era With the recent introduction of blockchain, "Nakamoto" consensus vi- olated the information-theoretic bounds no consensus for $t \geq n/3$, showing that even a majority of corrupted parties can be tolerated as long as the majority of the computation resources remain at honest hands, even without public key infrastructure. An intriguing question comes to mind, can the dolev and strong impossibility bounds, where they consider [LSP82] model (of instant delivery authenticated channels and full synchrony) be circumvented under the resource-restricting paradigm?

[Gar+20] answered this question in the affirmative, by representing resource-restricting paradigm as an access restriction on the underlying communication network.

Basically, the parties are restricted from sending unbounded messages to other parties. The paper con- structs the model in Canetti's Universal Composition framework [CA02], representing the restriction of the resources available to the adversary by means of a functionality wrapper, which wraps a communication network and restricts the ability of parties (or the adversary) to send new messages through this network. The paper only deals with synchronous setting.

6.0.1 Revisiting Impossibility Theorem

In [DLS88], it provided two proofs; one for which communication is partially synchronous, and one for which processors are partially synchronous. We attempt to propose a comparable proof where both processors and communication are partially synchronous.

Lemma

For the case n=3 and t=1, the proof is based on constructing three scenarios σ_1, σ_2, and σ_3.

In σ_1, all initial values are 0, the processor P_3 is initially dead, and the messages sent between P_1 and P_2 are delivered in exactly time 1. By t-resiliency, parties P_1 and P_2 decide 0 within some

finite time; say T_A.

In σ_2, P_2 is initially dead, and the messages sent between P_1 and P_3 are delivered in exactly time 1. By similar argument, P_1 and P_3 decide 1 within some finite time; say T_B.

In σ_3, P_2 has initial value 0, and P_3 has initial value 1, and P_1 is byzantine. P_1 behave with respect to P_2 like scenario σ_1, and with respect to P_3 like scenario σ_2. The messages sent between P_1 and P_2, as well as, P_1 and P_3 are delivered in exactly time 1, but all the messages between P_2 and P_3 are delivered within time $T_{AB} > 1$. Also, party P_3 has a clock-drift of -(T_A-1), in other words, it wakes up after time $T_A - 1$. Party P_2 (respectively P_3) acts exactly like in scenario σ_1 (respectively σ_2. Therefore, P_2 will decide 0 by time T_A, and party P_3 will decide 1 by time $T_A + T_B - 1$. Hence, arriving at contradiction.

Proof:

Theorem: Assume the model with byzantine fault, in which communication and and processors are partially synchronous. Assume,

$$pq < \frac{1}{(\Phi + \Delta) \cdot \gamma \cdot n} \tag{6.1}$$

where p is the probability of success of random oracle query, q is the number of queries $0 < q \leq 1$**,** Δ **is the upper bound on message delay and** Φ **is the upper bound on the clock drift. Then, the impossibility theorem from DLS doesn't go through.**

Proof. For simplicity, let $\Delta` = \Delta + \Phi$ and $pq = \frac{1}{(1+\alpha)(\Phi+\Delta)\cdot\gamma\cdot n}$.

In every fragmant of Δ' rounds, the number of messages sent by the adversary (number of successful random oracle queries) is distributed as binomial distribution with parameters $\Delta' \cdot q$ (trials) and p(success probability). Let random variable X_i be 1 if i-th query is successful, and 0 otherwise. Let $k\Delta'$ be the total number of rounds. We will show that the adversary can't send more

than $< k$ messages except with negligible probability. Let $X = \Sigma_{i=0}^{k\Delta'}\Sigma_{j=1}^{q}\Sigma_{k=1}^{\gamma\cdot n}$. It holds that $E[X] = k\Delta' \cdot q \cdot \gamma \cdot n \cdot p$. By an application of Chernoff bound, for $\delta = \alpha$ we obtain

$$\begin{aligned} Pr[X \geq (1+\delta)E[X] &= Pr[X \geq (1+\delta)k\Delta' q\gamma np] \\ &= Pr[X \geq \frac{k(1+\delta)}{(1+\alpha)}] \\ &= Pr[X \geq k] \\ &\leq e^{-\frac{\alpha^2}{2+\alpha}\frac{k}{(1+\alpha)}} \\ &\leq e^{-\Omega(\alpha k)} \end{aligned} \tag{6.2}$$

Hence, with negligible probability, the adversary will be able to send k or more messages in the $k\Delta'$ rounds. On the other hand, any party can send at most k-1 messages with overwhelming probability. Let $Y = \Sigma_{i=0}^{k\Delta'}\Sigma_{j=1}^{q}$.

Since,

$$Pr[Y < K] = 1 - Pr[Y \geq K] \tag{6.3}$$

We need to show that $Pr[Y \geq K]$ is negligible so that equation (3) would have overwhelming probability. It holds that $E[Y] = k\Delta' \cdot q \cdot p$. By an application of Chernoff bound for $\delta = \alpha\gamma n + \gamma n - 1$ we obtain

$$\begin{aligned} Pr[Y \geq (1+\delta)E[Y] &= Pr[Y \geq (1+\delta)k\Delta' qp] \\ &= Pr[X \geq \frac{k(1+\delta)}{(1+\alpha)\gamma n}] \\ &= Pr[X \geq k] \\ &\leq e^{-\Omega(\alpha k\gamma n)} \end{aligned} \tag{6.4}$$

7. Broadcast Primitive in Bounded-delay Network

The protocol uses the following broadcast primitive from [ST05]. Basically, for a party i to accept a message from party j, it has to receive at least 2t+1 (echo, j, m) messages. This helps in restricting the adversary from sending two conflicting messages to honest parties.

The broadcast primitive achieves the following properties:

1. Correctness: if p is correct, all correct processors agree on the value of its message.

2. Reliability: If p is faulty then, either all correct processor agree on the same value or none of them accepts.

Since we are assuming the unknown bounded model instead of GST, parties don't have to keep echoing the messages, resulting in lower communication complexity. This is due that messages can be lost before GST, and parties need to keep echoing the message after N-2t messages are received. No messages are lost in the unknown bounded model according to DLS definition.

Technical Remarks:

- Each party will wait for max(δ clock-ticks, before proceeding to the next round.

Broadcast primitive

- Party i sends (init,i,m) to all
- Code(party j)
  - if received (init,i,m), send (echo,i,m) to all
  - if received(echo,i,m) from N-2t distinct processors, send(echo,i,m) to all
  - if received(echo,i,m) from N-t distinct processors, accept m from party i.

Figure 7.1: Broadcast Primitive

- We will be using broadcast/accept instead of send/receive following the broadcast primitive mentioned earlier.

- A value v is acceptable to p if p doesn't have a lock on any value except possibly v.

Time Complexity: $O(\Delta \cdot log(\Delta)) + O(t)$

Protocol

Algorithm (P_j)

1. $\delta \leftarrow 1$
   //Round s: Propose
2. $k \leftarrow i$ mod n and $s \leftarrow 3k - 2$
3. set $\delta = 2 \cdot \delta$
4. Broadcast message $(list, k)$, where list contains acceptable values that are also in the proper list
5. if $j = i \; mod \; n$, wait to accept messages from $N - t$ parties that finds v acceptable. Else, choose a random value from its PROPER set and Broadcast $(lock, v, k)$
6. Start $timer_i$ [δ] if not yet done; wait until $timer_i$ [δ] has expires;

   //Round s+1: Acknowledge
7. If $j \neq i$ mod n, and has accepted $(lock, v, k)$ message and accepted N-t $(list, k)$ messages from parties, lock v and send acknowledgment (ack, k) to party P_i.
8. Release any earlier locks on value v
9. if $j = i \; modn$ and received acknowledgments from at least 2t+1 parties, decide v and Broadcast $(decide, v)$. Continue to participate in the protocol
10. Start $timer_i$ [δ] if not yet done; wait until $timer_i$ [δ] has expires;

    //Round s+2: Lock-release
11. If $j \neq i$ mod n and has a lock on same value v with associated phase h and has accepted, at this round or earlier, a valid lock on w with associated phase h', and if $w \neq v$, and $h' \geq h$, then P_j releases its lock on v.
12. if a party accepts t+1 $(decide, v)$ messages from different resources, it decides v and broadcasts $(decide, v)$.
13. if a party accepts 2t+1 $(decide, v)$, it halts and outputs $(decide, v)$
14. Start $timer_i$ [δ] if not yet done; wait until $timers_i$ [δ] has expires;
15. Go to first round

    Procedure: Weak Broadcast(m)
16. Party i sends $(init, i, m)$ to all
    // Code(party j)
17. if received $(init, i, m)$, send $(echo, i, m)$ to all
18. if received $(echo, i, m)$ from N-2t distinct processors,send $(echo, i, m)$ to all
19. if received $(echo, i, m)$ from N-t distinct processors, accept m from party i

Figure 7.2: Consensus Protocol in the Partial Synchronous setting

8.0.1 Redefining the UC Functionalities

In order to apply the resource restricted paradigm to the partial synchronous setting, we have to redefine the UC functionalities.

To formalize the bounded-delay on message delivery, a modified version of the authentication functionality from [Gar+20] will be used, however, the message will be delivered after a delay set by the adversary rather than after a single round.

For the bound on speed-drift between honest parties, we use the following version of imperfect local clocks from [Bad+19] that allow parties to proceed at roughly the same speed, where "roughly" is captured by the upper bound ϕ on the drift between any two honest parties. The only difference added to the functionality is that it returns the clock-value(whether correct or drifted) to the party, instead of returning the boolean variable $drift_p$. This facilitate the wrapper's task in blocking the messages according to the party's current round.

By restricting the underlying communication network, restricting the adversarial party from sending unboundedly many new messages than the other party, the impossibility results aren't applicable anymore. We achieve this with the help of a filtering wrapper functionality. The filtering wrapper restricts the per-round access of each party to the functionality. Each party has a quota of q send requests per round, and after a message has been sent through the filter, the sender, as well as the receiver can re-send the same message for free.

8.0.2 Graded Consensus in the partial Synchronous Setting

In [Gar+20], it used graded consensus by Feldman [FM88] to reach consensus on the parties' public keys. Since [FM88] only works in the synchronous setting, we had to design a graded

Verifiable Secret Sharing in Partial Synchronous Setting

Sharing Phase:

- Step 1: Dealer send shares
  1. The dealer choose bi-variate polynomial D such that D(0,0)=s
  2. For each party P_i, the dealer sends $D(x, \alpha_i) = F_i(x)$ and $D(\alpha_i, y) = G_i(y)$
- Step 2: Parties check inconsistencies (code for party P_i)
  1. For every party P_j, party P_i sends $F_i(\alpha_j)$ and $G_i(\alpha_j)$
  2. if $F_i(\alpha_j) = G_j(\alpha_i)$ and $G_i(\alpha_j) = F_j(\alpha_i)$, send (authenticate, i, j)
  3. If if $F_i(\alpha_j) \neq G_j(\alpha_i)$ and/or $G_i(\alpha_j) \neq F_j(\alpha_i)$, send complaint(i, j, $F_i(\alpha_j)$, $G_i(\alpha_j)$)
- Step 3: Resolve inconsistencies (Dealer's code)
  1. the dealer checks if $F_i(\alpha_j) = G_j(\alpha_i)$ and $G_i(\alpha_j) = F_j(\alpha_i)$, if yes, it does nothing.
     if no, it broadcasts reveal(i, $F_i(x), G_i(y)$)
- Step 4: Evaluate complaint resolutions (Code for party P_i):
  1. For every $j \neq k$, party P_i marks (j, k) as a joint complaint if it viewed two messages complaint(k, j, u1, v1) and complaint(j, k, u2, v2) broadcast by P_k and P_j , respectively, such that $u_1 \neq v_2$ or $v_1 \neq u_2$.
  2. Consider the set of $reveal(j, f_j(x), g_j(y))$ messages sent by the dealer: If there exists a message in the set with $j \neq i$ and for which $f_i(\alpha_j) \neq g_j(\alpha_i)$ or $g_i(\alpha_j) \neq f_j(\alpha_i)$, then output complain(i,j,$f_i(\alpha_j), g_i(\alpha_j)$, and go to last step .
     If there exists a message in the set with j = i then reset the stored polynomials $f_i(x)$ and $g_i(y)$ to the new polynomials that were received, and go to the last Step (without broadcasting consistent).
- Step 5: output decision:
  1. Wait until all joint complaints (j, k) are resolved for which the dealer broadcast $reveal(k, fk(x), gk(y))$ or $reveal(j, fj(x), gj(y))$ and that the new polynomials satisfy party $P_i's$ polynomials: If pairwise authenticated 2t+1 parties, broadcast (consistent)
  2. wait until the receive of 2t+1 consistent messages, if P_i sent consistent, set $s_i = F_i(x)$, else set $S_i = \perp$

Core-Set Protocol:

- Run n invocation of Byzantine Agreement for each P_j. If BA(P_i) =1. Add P_i to the core-set C
- If lCl = n-t, start the reconstruction phase.

Reconstruction Phase:

1. Every party P_j in the core-set C, broadcast $F_j(x)$
2. Each party P_i needs to validate the $F_j(x)$ with the points shared earlier, if so, broadcast verify(i, j)
3. Wait until the receive of n-t verification messages(*, j), add $F_j(0)$ to the interpolation set I.
4. Wait until lIl = t+1 and Interpolate the secret s, broadcast(secret, s)
5. wait until the receive of 2t+1(secrect, s), terminate.

Figure 8.1: Verifiable Secret Sharing

Verifiable Secret Sharing in Partial Synchronous Setting

Sharing Phase:

- Step 1: Dealer send shares
  1. The dealer choose bi-variate polynomial D such that D(0,0)=s
  2. For each party P_i, the dealer sends $D(x, \alpha_i) = F_i(x)$ and $D(\alpha_i, y) = G_i(y)$
- Step 2: Parties check inconsistencies (code for party P_i)
  1. For every party P_j, party P_i sends $F_i(\alpha_j)$ and $G_i(\alpha_j)$
  2. If if $F_i(\alpha_j) \neq G_j(\alpha_i)$ and/or $G_i(\alpha_j) \neq F_j(\alpha_i)$, A-cast (Complaint, i, j, $F_i(\alpha_j)$, $G_i(\alpha_j)$)
- Step 3: Resolve inconsistencies (Dealer's code)
  1. the dealer checks if $F_i(\alpha_j) = G_j(\alpha_i)$ and $G_i(\alpha_j) = F_j(\alpha_i)$, if yes, it does nothing.
     if no, it A-casts (Reveal, i, $F_i(x), G_i(y)$)
- Step 4: Evaluate complaint resolutions (Code for party P_i):
  1. For every $j \neq k$, party P_i marks (j, k) as a joint complaint if it viewed two messages $(Complaint, k, j, u1, v1)$ and $(Complaint, j, k, u2, v2)$ A-casts by P_k and P_j , respectively, such that $u_1 \neq v_2$ or $v_1 \neq u_2$.
  2. Consider the set of $(Reveal, j, f_j(x), g_j(y))$ messages sent by the dealer: If there exists a message in the set with $j \neq i$ and for which $f_i(\alpha_j) \neq g_j(\alpha_i)$ or $g_i(\alpha_j) \neq f_j(\alpha_i)$, then output $(Complaint, i, j, \mathrm{f_i}(\alpha_j), g_i(\alpha_j)$.
     If there exists a message in the set with j = i then reset the stored polynomials $f_i(x)$ and $g_i(y)$ to the new polynomials that were received.
- Step 5: Core-Set Protocol:
  - Run n invocation of for each P_j. If (P_j) =1. Add P_j to the core-set C
  - If lCl = N-t, start the reconstruction phase.

Reconstruction Phase:

1. Every party P_i in the core-set C, A-casts $F_i(x)$
2. Each party P_j needs to validate the $F_j(x)$ with the points shared earlier
3. Run n invocations of on party P_i's input.
4. If $(F_i(x))$ terminates and it aligns with the points sent earlier, add P_i to set lIl.
5. Wait until lIl = N-2t and Interpolate the secret s, output(secret, s)

Figure 8.2: Verifiable Secret Sharing in bounded-delay

$\mathcal{F}_{\text{AUTH}}$ $^{\Delta}$

The functionality registers with the global clock $Gclock^{\phi}$, and is parameterized by a set of possible senders and receivers, denoted by P, a list M', and integer variables of the form D_z, where z $\in \{0,1\}^*$, that are dynamically created. For every party p $\in$ P it maintains a fetch counter f_p . Initially, $M' = \emptyset$ and $f_p = 0$, for every P $\in$ P.

- Upon receiving (send,sid, m, Pj) from Pi $\in$ P, it sends (Clock-get, sidC, pj) to $Gclock^{\phi}$. it sets $D_{mid} := 1$, $D^{max}_{mid} = 1$ and $M' = M'||(m, Pi, Pj, mid, \tau_{sid}, clock_{pj})$, where mid is a unique message-ID, and send (sent, sid, m, Pi , Pj , mid) to A.
- Upon receiving (delays, mid, d) from the adversary, for each pair(mid, d) do 1. If D^{max}_{mid} + d $\leq \triangle$, then set $D_{mid} = D_{mid} + d$ and $D^{max}_{mid} = D^{max}_{mid} + d$ and return (delay-set) to the adversary, otherwise, ignore message.
- Upon receiving (fetch,sid) from some honest party Pj $\in$ P, increment f_p by 1, set M = $\emptyset$, and do the following:
  1. For all tuples (m, Pi , Pj , mid) $\in M'$, set $D_{mid} := D_{mid} - 1$
     - sends (Clock-get, sidC, pj) to $Gclock^{\phi}$, and update "$clock_{pj}$"
  2. For all tuples (m, Pi , Pj , mid) $\in M'$, where $D_{mid} \leq 0$, and τ_{sid} = $clock_{p_j}$, delete (m, Pi , Pj , mid) from M' , and add (m, Pi) to M.
  3. Send (sent, sid, mid) to Pj .
- Upon receiving (fetch-requests,sid, P) from A, output (fetch-requests,sid, f_P)

Figure 8.3: Authentication Functionality

consensus protocol that works in the partial synchronous setting. The protocol uses the following broadcast primitive from [ST05]. Basically, for a party i to accept a message from party j, it has to receive at least 2t+1 (echo, j, m) messages. This helps in restricting the adversary from sending two conflicting messages to honest parties.

Weak Broadcast Primitive

- Party i sends $(init, i, m)$ to all
- Code (party j)
  - if received $(init, i, m)$, send $(echo, i, m)$ to all
  - if received $(echo, i, m)$ from N-2t distinct processors, send $(echo, i, m)$ to all
  - if received $(echo, i, m)$ from N-t distinct processors, accept m from party i.

The broadcast primitive achieves the following properties:

$\bar{\mathcal{G}}_{\text{CLOCK}}{}^{\phi}$

The functionality manages the set P of registered identities, i.e, parties P = (pid,sid). It also manages the set F of registered functionalities (together with their session identifier). Initially, $P = \emptyset$ and $F = \emptyset$.

For each session sid, the clock maintains a variable τsid

For each identity $P := (pid, sid) \in P$ it manages bit variables d_p and d_{imp}, and an integer $drift_P$. For each pair $(F, sid) \in F$ it manages variable $d(F, sid) \in 0, 1$ (all these variables are initially set to 0).

Synchronization:

- Upon receiving (clock-update, sidC) from some party $P \in P$ set d_p^{imp}:= 1 and d_p := 1; execute Round-Update and forward (clock-update, sidC , P) to A.
- Upon receiving (clock-update, sidC) from some functionality F in a session sid such that (F, sid) ∈ F set d(F,sid) := 1, execute Round-Update and return (clock-update, sidC , F) to this instance of F.
- Upon receiving (clock-push, sidC , P) from A where party $p \in P$, if d_p^{imp}:= 1 and $drift_P < \phi$ clock then update d_p^{imp}:= 0 and $drift_P$:= $drift_P$ + 1 and return (clock-push-ok, sidC , P) to A. Otherwise ignore the message.
- Upon receiving (clock-get, sidC, p_i) from any participant P—including the environment on behalf of a party—or the adversary on behalf of a corrupted party P (resp. from any ideal—shared or local—functionality F), execute procedure Round-Update; return (clock-get, sidC, τ + $drift_p$) (resp. (clock-get, sidC , d(F,sid)))) to the requester (where sid is the sid of the calling instance).

Procedure Round-Update: For each session sid do: If d(F,sid) $= 1$ for all $f \in F$ and $d_p = 1$ for all honest parties P = (·, sid) ∈ P, then update $d(F, sid) = 0$, update $d_p = 0$ and $drift_P = drift_P - 1$ for all parties, and set $\tau sid = \tau sid + 1$ P = (·, sid) ∈ P; for all P = (·, sid) ∈ P with $drift_P < 0$ reset d_p^{imp} := 0 and $drift_P$:= 0.

Figure 8.4: Global Clock Functionality

Wrapper Functionality $W_{FLT}^{p,q}(F)$

The wrapper functionality is parameterized by p ∈ [0, 1] and q , which restrict the probability of success and number of F-evaluations of each party per round, respectively, and a set of parties P. Assuming that $0 < q \leq 1$, set q = 1/q. The functionality registers with the global clock $Gclock^{\phi}$. It manages the round integer variables τ_i for each party, the current set of corrupted parties P, and a list T . For each party P ∈ P, it manages the integer variables t_p and f_p . Initially $\tau = 0$, $T = \emptyset$, and $t_p = 0$, for each p ∈ P.
Filtering:
Upon receiving (send, sid, m, Pj) from party $p_i \in$ P, execute Round-Reset, and do the following:

- – If $t_{pi} \geq$ q, with probability p, do:
  1. set $t_{pi} = 0$
  2. Add (m, P_i) to T and output (success, sid) to P_i
  3. On response (continue, sid, m) from P_i , forward (send, sid, m, Pj) to F. In any other case, send (fail, sid) to Pi.
- Upon receiving (resend, sid, m, Pj) from honest party $Pi \in P$ P', if (m, Pi) ∈ T then forward (send, sid, m, Pj) to F.
- Upon receiving (resend,sid, m, Pj) from A on behalf of corrupted $pi \in P'$, if (m, Pi) ∈ T for some P ∈ P, then forward (send,sid, m, Pj) to F.
- Upon receiving (fetch,sid) from some party Pj ∈ P, if fp := 0, set fp :=1 and forward the message to Fauth. Else, do nothing.
- Upon receiving (delay, mid, d) from the adversary, forward the message to Fauth.
- Upon F sending (sent,sid, m) to Pj , forward the message to Pj .

Standard UC Corruption Handling:
-Upon receiving (corrupt, sid, P) from the adversary, $setP' = P' \bigcup P$.
General:
-Any other request from (resp. towards) any participant or the adversary, is simply relayed to the underlying functionality (resp . any participant of the adversary) without any further action.
Procedure Round-Reset:

- Send (clock-Get, sidC, p_i) to $Gclock^{\phi}$ and receive (clock-Get, sidC , τ_{pi}) from $Gclock^{\phi}$
- If τ' - $\tau \geq 0$, then set $t_{pi} := t_{pi} + \tau' - \tau$and fp := 0 for each pi ∈ P and set $\tau = \tau'$.

Figure 8.5: Wrapper Functionality

1. Correctness: if p is correct, all correct processors agree on the value of its message.

2. Reliability: If p is faulty then, either all correct processor agree on the same value or none of them accepts.

Since we are assuming the unknown bounded model instead of GST, parties don't have to keep echoing the messages, resulting in lower communication complexity. This is due that messages can be lost before GST, and parties need to keep echoing the message after N-2t messages are received. No messages are lost in the unkown bounded model according to DLS definition.

The Graded Consensus Protocol definition follows from [Fit02]

Graded Consensus Protocol: Let P be a jointly terminating protocol. Each good player i outputs a pair $(code_i, value_i)$, where $code_i$ is 0, 1. We say that P is a Graded-Consensus if the following conditions are satisfied:

1. Validity: If all correct players hold the same input value v, all correct players will decide on v and output $code_i = 1$

2. Consistency: If any correct player i outputs $code_i = 1$, then players j decide on the same output value, $value_i = value_j$.

Technical Remarks:

- Assume v to be the value set by party i, v' to be 1-v.

- "distinct" means different messages than the party's own messages.

- We will be using broadcast/accept instead of send/receive following the broadcast primitive mentioned earlier.

- In step 2, messages are numbered. For a party to accept(decide,v, 1) from party j, it must

Grade-cast

Common Parameters:

- n, the size of the network
- t, an upper bound on the number of bad players($t < n/3$)
- v_i, the value set by each party

Code (For every player i)

1. broadcast (init, v_i) to all processors
2. Wait until the receipt of n-t acceptable messages.
   (a) Set v_i to the majority of the messages received.
   (b) broadcast (decide, v_i, 0*) to all parties
3. Wait until the receipt of
   (a) 2t+1 (decide, v_i, 0*) acceptable messages, output $(v_i, 1)$ and halt.
   (b) 2t+1 (decide, v_i, 0*) with at least one (decide, v_i, 1*) acceptable messages, output (v_i,0)
   (c) distinct t+1 (decide, v_i', 0*) acceptable messages, broadcast (decide,v_i', 1)

Figure 8.6: Gradecast

have accepted (decide,v,0) from the same party.

Lemma 8.0.1. *If all parties start with the same value v, no party will send (decide, v',0).*

Proof. In step 2, honest parties wait to receive n-t acceptable messages before setting v_i to the majority of the messages received, and then sending (decide, $v_i, 0$). Sine $n - t \geq 2t + 1$, the party has to receive at least acceptable t+1 messages of the same v to adopt this value. since there are are most t malicious parties and honest parties can't send conflicting messages, all honest parties will send (decide,v,0). □

Lemma 8.0.2. *If all honest players start with the same value v, every good player i outputs (v,1).*

Proof. From lemma 0.1, if all honest parties start with the same value, all honest parties will send (decide,v, 0). Since there are at most t malicious parties, no honest party will ever receive distinct t+1 acceptable messages (decide,v', 0). Also, since a party has to accept (decide,v, 0) before accepting (decide,v,1), the adversary can't send (decide,v,1) to force the honest parties to output

(v,0). Eventually, all honest parties will receive 2t+1 acceptable (decide, v, 0) and output (v,1). □

Lemma 8.0.3. *If any honest player outputs (v,1), all other honest players will output (v,*)*

Proof. For an honest player to output (v,1), he has to receive at least 2t+1 acceptable (decide, v, 0) messages. Out of the 2t+1 messages, t is at most malicious. Therefore, every honest party i will eventually receive t+1 acceptable (decide,v, 0). If i started with the same value v, it will halt with output (v,1)/(v,0). If it started with v'(or sent (decide,v',0)), it will send (decide,v,1) and halt with (v,0).

In other words, based on the broadcast primitive, a faulty party can't send two acceptable conflicting messages to parties; he can only send either (decide,v, 0/1) or (decide,v',0/1) to all parties. If the parties started with the same value, the adversary's messages won't affect anything, and all parties will halt with output (v,1) from lemma 0.2. If the honest parties didn't start with the same value, assuming n=4 for simplicity, two honest parties will send (decide,v,0) and one honest party will send (decide, v',0) message, one of the following scenarios could happen

1. If the adversary sends (decide,v,0), the two honest parties will output (v,1). The other party that sent (decide,v',0), will eventually receive t+1 acceptable (decide, 1-v') from the honest parties or the adversary and output (v,0)

2. If the adversary sends (decide,v'), parties will output different (v'/v,0). However, none will output (*,1) as it needs to receive 2t+1 acceptable (decide,*,0) messages before, which can never happen.

3. If the adversary doesn't send anything, the single honest party with (decide,v',0) will eventually receive t+1 acceptable (decide, 1-v',0) messages from the two honest parties and will send (decide, v,1) and output (v,0). The other two will output (v,0).

□

Theorem 8.0.4. *The graded consensus protocol is a t-resilient protocol, for t<n/3*

Proof. From lemma 0.1, 0.2, 0.2 0.4, the protocol achieves Fitzi's definition for graded consensus; achieving validity and agreement. □

REFERENCES

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. "The Byzantine Generals Problem". In: *ACM Trans. Program. Lang. Syst.* 4.3 (July 1982), pp. 382–401. ISSN: 0164-0925. DOI: 10.1145/357172.357176. URL: https://doi.org/10.1145/357172.357176.

[DS83] Danny Dolev and H. Strong. "Authenticated Algorithms for Byzantine Agreement". In: *SIAM J. Comput.* 12 (Nov. 1983), pp. 656–666. DOI: 10.1137/0212045.

[FLP85] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. "Impossibility of Distributed Consensus with One Faulty Process". In: *J. ACM* 32.2 (Apr. 1985), pp. 374–382. ISSN: 0004-5411. DOI: 10.1145/3149.214121. URL: https://doi.org/10.1145/3149.214121.

[DLS88] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. "Consensus in the Presence of Partial Synchrony". In: *J. ACM* 35.2 (Apr. 1988), pp. 288–323. ISSN: 0004-5411. DOI: 10.1145/42282.42283. URL: https://doi.org/10.1145/42282.42283.

[FM88] Paul Feldman and Silvio Micali. "Optimal Algorithms for Byzantine Agreement". In: *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*. STOC '88. Chicago, Illinois, USA: Association for Computing Machinery, 1988, pp. 148–161. ISBN: 0897912640. DOI: 10.1145/62212.62225. URL: https://doi.org/10.1145/62212.62225.

[Fit02] Matthias Fitzi. "Generalized communication and security models in Byzantine agreement". In: 2002.

[ST05] T. K. Srikanth and Sam Toueg. "Simulating authenticated broadcasts to derive simple fault-tolerant algorithms". In: *Distributed Computing* 2 (2005), pp. 80–94.

[Nak09] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. 2009. URL: http://www.bitcoin.org/bitcoin.pdf.

[Bad+19] Christian Badertscher et al. "Ouroboros Chronos: Permissionless Clock Synchronization via Proof-of-Stake". In: *IACR Cryptol. ePrint Arch.* 2019 (2019), p. 838.

[Gar+20] Juan A. Garay et al. "Resource-Restricted Cryptography: Revisiting MPC Bounds in the Proof-of-Work Era". In: *Advances in Cryptology – EUROCRYPT 2020* 12106 (2020), pp. 129–158.