CMSC 858K — Advanced Topics in Cryptography	May 4, 2004
Lecture 26	
Lecturer: Chiu Yuen Koo	*Scribe(s): (none)*

1 Introduction

In this lecture, we study the Byzantine Agreement problem, defined as follows: consider a network of n processors, where each pair of processors can communicate (this is the so-called "point-to-point" model). Furthermore, at most t processors within this network may be faulty; a faulty processor may exhibit arbitrary behavior. (We also assume that the behavior of these faulty processors may be coordinated by a single adversary, and sometimes do not place any computational restrictions on this adversary.) Initially, each processor has an input value p_i; this group of processors then runs a protocol which results in each (non-faulty) processor deciding on a value p_i^*. Besides requiring that the protocol terminate, a *Byzantine agreement* protocol also satisfies the following (as long as no more than t processors are faulty):

Agreement All non-faulty processors decide on the same value. I.e., if i, j are non-faulty then $p_i^* = p_j^*$.

Validity If the initial inputs of all non-faulty players are identical, then all non-faulty players decide on that value. I.e., if $p_i = p^*$ for all non-faulty players i, then $p_i^* = p^*$ for all non-faulty players i.

We remark that either one of these properties is trivial to achieve on their own (agreement by having all non-faulty processors always output "0", and validity by having each processor i output $p_i^* = p_i$); the tricky part is to guarantee that they both hold in tandem.

The Byzantine agreement problem was first formulated by Lamport, Pease, and Shostak [5, 3], and was motivated by fault-tolerant systems where a set of independent processors are required to come to an exact agreement in presence of faulty processors. Examples include synchronization of internal clocks or agreement on sensor readings. From a cryptographic perspective, Byzantine agreement is a central primitive used within multi-party computation protocols, where now a certain fraction of adversarial processors may be actively trying to prevent agreement among the honest processors.

1.1 Broadcast

A broadcast channel is (as one might expect) a channel to which any player may send a value which is then received by all other players (we also assume that players can tell which player sent the value). Note that if a broadcast channel is available, then the Byzantine agreement problem is trivial (for $t < n/2$): each player simply broadcasts their value p_i and then honest players decide on the majority value. Unfortunately, most real-world systems (at best) only guarantee "point-to-point" communication as we have described above.

However, it is worthwhile to consider broadcast as a *functionality* (rather than as simply an atomic primitive); doing so, we come up with the following definition: Assume an n-party network with point-to-point channels, as above. We now assume a distinguished party s within this network, called *the sender*, who initially holds an input value p_s. As before, we will allow up to t parties within the network (the dealer possibly included) to be faulty. The processors in the network then run a protocol which results in each non-faulty player deciding on a value p_i^*. In addition to requiring that the protocol terminate, a *broadcast* protocol also satisfies the following (as long as no more than t players in total are faulty):

Agreement All non-faulty players decide on the same value; i.e., $p_i^* = p_j^*$ for all non-faulty i, j.

Correctness If the dealer is honest, then all non-faulty players decide on p_s. I.e., if the dealer is non-faulty then $p_i^* = p_s$ for all non-faulty players i.

(Note that broadcast may now be implemented by a *protocol*, rather than only by an atomic "broadcast channel".) We have noted above that if we can achieve broadcast and $t < n/2$, then we can achieve Byzantine agreement. In fact, the opposite direction also holds (for any t, although as defined above Byzantine agreement is not meaningful when $t \geq n/2$), as the following protocol shows: first, the dealer sends his value p_s to each other player (using a point-to-point channel). Next, the players run a Byzantine agreement protocol and decide on the result. It is not hard to see that agreement and correctness both hold.

Because of this equivalence, we are free to focus on either Byzantine agreement or broadcast. In the remainder of this lecture, we focus on broadcast. A key results in this area is the following, which we will prove in this and the next lecture:

Theorem 1 *Broadcast (or Byzantine agreement) is possible iff $t < n/3$ (or $t \geq n - 1$, which is a trivial case). The possibility result holds even for computationally-unbounded adversaries who coordinate the actions of all faulty players. The impossibility result holds even for computationally-bounded adversaries (as long as the adversary is allowed to run for essentially the same amount of time as honest processors), assuming no prior set-up phase is allowed.*

This result was first proved in [5]. A few comments are in order:

1. We have not yet said anything about protocol efficiency, and indeed the original protocol for Byzantine agreement (when $t < n/3$) ran in exponential time. Of course, we ultimately want a protocol that runs in polynomial time. Since the initial work of [5, 3], much further work has focused on constructing polynomial-time protocols, and then optimizing the number of rounds/messages/etc.

2. The remark at the end of the theorem will be discussed later in the lecture. As a preview, note that one possibility for a set-up phase is to assume a PKI such that each player i (including the faulty ones) has established a public/secret key pair (PK_i, SK_i) for a digital signature scheme, with the same value PK_i known to all other players.

3. Other work in this area has focused on what can be achieved within different communication models (say, where a channel does not necessarily exist between every pair of players, etc.).

2 The Impossibility Result

In this section, we prove the "only if" part of the theorem; i.e., we prove:

Theorem 2 *Broadcast (and hence Byzantine agreement) is not achievable when the number of faulty player t satisfies $n - 1 > t \geq n/3$. (Note that $t \geq n - 1$ implies that the number of honest players is at most 1, in which case the whole problem becomes trivial.)*

Proof We first focus on the case $n = 3$, $t = 1$, and then prove the case for general n. Call the sender s and label the two other players "1" and "2". In the real network, each of these players is connected with the other two. But imagine now that we make copies $s', 1', 2'$ of each of these players and arrange them in a hexagon as follows:

$$\begin{array}{ccccc} s & \leftrightarrow & 1 & \leftrightarrow & 2 \\ \updownarrow & & & & \updownarrow \\ 2' & \leftrightarrow & 1' & \leftrightarrow & s' \end{array}$$

Now, assume toward a contradiction that we have some (possibly randomized) protocol P for broadcast. This protocol completely specifies what each player $s, 1, 2$ should do given its initial input (in the case of s) and the messages it receives from the other parties. Consider the case when the players above execute the protocol honestly, communicating as in the diagram, *but where s has input 0 and s' has input 1.* We claim the following:

1. Let us look at things from the point of view of players 1 and 2. The key is to realize that, from their perspective, the combined actions of s and s' represent possible adversarial activity of the sender s in the original network. (In particular, the "real" sender s in the original, 3-player network can simulate the actions of $1'$, $2'$, s with input 0, and s' with input 1.) Furthermore, from this viewpoint players 1 and 2 are acting completely honestly. Since, by assumption, P is a protocol for broadcast, we must then have that 1 and 2 output the same value. Letting p_1 (resp., p_2) represent the output of player 1 (resp., 2), we then have $p_1 = p_2$.

2. Now, look at things from the point of view of players s and 1. Here, the key is to realize that, from their perspective, the combined actions of players 2 and $2'$ represent possible malicious activity of player 2 in the original network. (Again, a malicious player 2 in the "real", 3-player network can simulate the actions of $2, 2', s, s'$.) But, again, players s and 1 are acting completely honestly. Since P is a broadcast protocol, it must be the case that player 1 outputs the initial input value of s. That is, $p_1 = 0$.

3. However, an exactly symmetric argument with respect to $s', 2$ shows that player 2 must output the initial input of s'; i.e., $p_2 = 1$.

The above give the desired contradiction (namely, we require $p_1 = p_2$ but $p_1 = 0$ while $p_2 = 1$), showing that the claimed broadcast protocol cannot exist.

We now prove the claim for the case of n a multiple of 3 (although a small modification of what we say extends to give a proof for arbitrary n). We do so by reducing in to the case $n = 3$. Namely, we show that if there exists a broadcast protocol for $n \leq 3t$ then we can construct a broadcast protocol for $n = 3$, $t = 1$.

So, assume we have a broadcast protocol P for n players (n a multiple of 3) and secure against t malicious parties, with $t \geq n/3$. We assume for simplicity that player 1 is the sender. Construct broadcast protocol P' for $n' = 3$ players $1', 2', 3'$ as follows (again, player $1'$ will be the sender): The basic idea is that player $1'$ will simulate players 1 through $n/3$; player $2'$ will simulate players $n/3 + 1$ through $2n/3$; and player $3'$ will simulate players $2n/3 + 1$ through n. In more detail, focusing on player $1'$ (actions of players $2'$ and $3'$ are defined similarly): If player $1'$ has input b, it runs player 1 (internally) with input b. Whenever players $i, j \in [1, n/3]$ in protocol P want to send a message to each other, player $1'$ simply simulates this internally. When player $i \in [1, n/3]$ wants to send a message m to player $j \in [n/3 + 1, 2n/3]$, player $1'$ sends the message (i, j, m) to player $2'$. When $j \in [2n/3+1, n]$, player $1'$ sends a similar message to player $3'$. When player $1'$ receives a message (j, i, m) from player $2'$ with $j \in [2n/3, n]$, player $1'$ internally passes message m to (internal) player i "from" player j. When such a message comes from player $3'$, player $1'$ acts appropriately. Players $2'$ and $3'$ output whatever is output (internally) by any of the players they are simulating.

Assume P is secure for $t \geq n/3$. We then claim that P' is secure for $t = 1$. To see this, note that the actions of any one compromised player in protocol P' can be simulated by the compromise of at most $n/3$ players in protocol P (namely, the $n/3$ players in P assigned to the corrupted player in P'). Consider the case when $1'$ is compromised in P', and hence players 1 through $n/3$ are compromised in P. Since agreement holds for P, we know that players $n/3 + 1$ through n all output the same value, and so players $2'$ and $3'$ will output the same value and agreement holds for P'. Next, consider the case when $2'$ is compromised in P', corresponding to compromise of players $n/3 + 1$ through $2n/3$ in P (the situation is analogous when player $3'$ is compromised). Since correctness holds for P, we know that players $2n/3+1$ through n output the initial input value of player 1. Since this latter value is the same as the initial input value of player $1'$, we have that player $3'$ outputs the initial input value of player $1'$ and hence correctness holds for P'. ■

In the next lecture, we will show a protocol for Byzantine agreement/broadcast for n players and $t < n/3$ faulty players.

3 "Authenticated" Broadcast

We now return to the remark about circumventing the impossibility result proved above if we allow an initial "set-up" phase. In particular, we will assume a PKI has been established such that each player i has a public-/secret-key pair (PK_i, SK_i) and (i, PK_i) is known to all other players. (We stress that *every* player holds the same PK_i for player i—this is crucial.) We will also assume a computationally-bounded adversary who cannot forge a signature (with non-negligible probability) on any previously-unsigned message with respect to the public key of any of the honest players.

It is worthwhile to see where the proof of the impossibility result from the previous

section breaks down in this case. Looking at the case $n = 3$, $t = 1$, we see that a crucial element in the proof is the ability of corrupted players to simulate the actions of non-corrupted players (for example, we required that a malicious s can simulate the actions of "honest" $1'$ and $2'$). But when the players might sign messages (and we assume a computationally-bounded adversary) *it is no longer necessarily possible for corrupted players to simulate the actions of honest players* and the proof break down.

We now describe the Dolev-Strong protocol [1] for "authenticated" broadcast when a PKI is established. Our description of the protocol is based on [2]. Again, we assume that player 1 is the sender. We first introduce some notation: a message received by party j in round i is called (v, i)-*authentic for* j if it has the form $(v, p_1, \sigma_1, \ldots, p_i, \sigma_i)$ where $v \in \{0, 1\}$, $p_1 = 1$, all p_i are distinct, $j \notin \{p_1, \ldots, p_i\}$, and, for all i, the signature σ_i is a valid signature with respect to PK_{p_i} of the string $(v, p_1, \sigma_1, \ldots, p_{i-1}, \sigma_{i-1})$. When j is clear from the context, we simply call this a (v, i)-authentic message. The protocol proceeds as follows:

Round 1: Party 1 signs its input v and sends $(v, 1, \sigma_1)$ to all parties. (Note that when player 1 is honest, this message is $(v, 1)$-authentic for all $j \neq 1$.)

Rounds 2 through $n - 1$**:** Each player j acts as follows in round i: for each $v \in \{0, 1\}$, if player j has received a $(v, i-1)$-authentic message $(v, p_1, \sigma_{p_1}, \ldots, p_{i-1}, \sigma_{i-1})$ in the previous round, then it signs this message and sends $(v, p_1, \sigma_{p_1}, \ldots, p_{i-1}, \sigma_{i-1}, j, \sigma_j)$ to all other players.

(Each player sends at most 2 messages per round to all players—one for each possible value of v—even if it has received many different $(v, i-1)$-authentic messages.)

Conclusion: Each party j decides on its output as follows:

If party j has ever received both a $(0, i)$-authentic message and a $(1, i')$-authentic message, for any $i, i' \leq n-1$, then it outputs the default value 0. (Note that when this occurs the sender must be cheating [assuming the security of the signature scheme], since it has issued signatures on two different values.)

If player j has ever received a (v, i)-authentic message for some i, but never received a $(\bar{v}, i')$-authentic message for any i', then it outputs v.

If player j has never received a (v, i)-authentic message for any value of v, then it outputs the default value 0. (Again, this situation implies that the sender must be dishonest, since it was supposed to send an authentic message in round 1.)

We claim that the above is a secure broadcast protocol. Correctness is easy to see: if the sender is honest and has initial input v, then every player receives a $(v, 1)$-authentic message; assuming the security of the signature scheme, players will never receive a $(\bar{v}, i)$-authentic message (since, in particular, this would require forging the signature of player 1) and so all honest players will output v.

For agreement, consider some honest party j and assume j received a (v, i)-authentic message in some round $i \leq n - 1$. There are two cases. If $i < n - 1$ then we claim that by the following round, every honest party k will have received a (v, i')-authentic message for some $i' \leq i + 1$. To see this, note that there are two sub-cases here: either k's signature

is already included in the (v, i)-authentic message received by j or not. In the first case, $k$must have already received a (v, i')-authentic message for some $i' < i$ (since otherwise it would not have produced the signature contaned in the (v, i)-authentic message received by j). In the second case, k will receive a $(v, i + 1)$-authentic message from j in the following round.

The other case is that $i = n - 1$ and so j received a $(v, n - 1)$-authentic message in the final round. In this case, the fact that this message is $(v, n - 1)$-authentic means that every other player has signed this message, and so every honest player has already received a (v, i)-authentic message in some previous round (again, assuming the security of the signature scheme so that signatures of honest players cannot be forged).

Putting this together, we have that if *any* honest player has received a (v, i)-authentic message by the end of the protocol, then *every* honest player has received a (v, i')-authentic message by the end of the protocol. It is then clear that agreement holds.

References

[1] D. Dolev and H.R. Strong. Authenticated algorithms for Byzantine agreement. *SIAM J. Computing* 12(4):656–666, 1983.

[2] O. Goldreich. *Foundations of Cryptography, vol. 2: Basic Applications*. Cambridge University Press, 2004.

[3] L. Lamport, R. Shostak, and M. Pease. The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst.*, 4(3): 382–401 (1982).

[4] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, Inc., 1996.

[5] M. Pease, R. Shostak, and L. Lamport. Reaching Agreement in the Presence of Faults. *J. ACM*, 27(2): 228–234 (1980).