In the first lecture I said that cryptography is the science of secure communication and computation in insecure environments. The ideal secure environment is one in which the whole infrastructure is managed by a trusted central authority. Cryptography is about eliminating this trust in authority by distributing it among the end users. Does it succeed in this?

Let's look at secure communication again. In an ideal environment, the central authority takes Alice's message and delivers it to Bob, who can trust that what he received is really Alice's message and that nobody else saw it. In private-key cryptography the central authority is replaced by a shared secret key. How are they supposed to share a key in an insecure environment? They could use public-key cryptography instead, so that all Alice needs to know is Bob's public key. If the environment is insecure, how can Alice be certain that what is claimed to be Bob's public key is really Bob's public key and not say Eve's?

In large-scale deployments of public key cryptography there is usually a database that associates the users' identities with their secret keys. Who should be allowed to manage this database? On the web there are several databases managed by various certificate authorities, and the web browser decides which of them are to be trusted and which are not. The integrity of today's public key infrastructure relies on a small number of influential authorities.

To reclaim trust the users need a mechanism to agree on a common reference like the contents of a public key database. Let's start by getting them to agree on anything.

1 Byzantine agreement

Byzantine agreement is the following rudimentary agreement problem. There is some number of parties each holding an input. The parties communicate via private point-to-point channels. Some parties may be corrupt and deviate from the protocol in any way they like. Each honest party needs to produce an output such that

1. All the (honest) parties' outputs are the same, and
2. If all the (honest) parties' inputs are the same then their outputs and inputs are equal.

Let's simplify things even more and assume each party's input is a bit value 0 or 1.

Here is a natural protocol for the three-party case: Each party announces its input and takes the majority of the announcements it has seen (including its own) as its output. If everyone is honest the outputs are all equal to the majority value of the inputs so both correctness conditions are satisfied.

What if Alice and Bob are honest but Charlie is not? Suppose Alice's input is 0 and Bob's input is 1. If Charlie tells Alice that his input is 0 but tells Bob that his input is 1, then Alice's and Bob's outputs will be different violating correctness.

We'll come back to three-party agreement shortly. Let's first describe a protocol for four parties. The key component is a protocol for a different problem in which the four parties are not symmetric. In the *Byzantine Generals Problem*, one party is a General and the other three are Lieutenants. The General proposes its input value to the Lieutenants. There are two functionality requirements:

1. The honest Lieutenants' outputs must always be the same.
2. Moreover, if the General is honest, their outputs must equal the General's proposal.

Four-party Byzantine Generals Protocol:

1. The General announces his proposal to the Lieutenants.
2. The Lieutenants forward the General's announcement to one another.
3. Each Lieutenant outputs the majority of the three received announcements.

Claim 1. *The four-party Byzantine Generals Protocol is secure against one corrupt party.*

Proof. If a Lieutenant is corrupt he won't affect the majority of the announcements, so they all equal the General's proposal. If the General is corrupt, each Lieutenant sees the same three announcements so all outputs are the same. □

Four-party Lamport-Shostak-Pease Protocol:

1. The Byzantine Generals Protocol is run three times, with Alice, Bob, and Charlie alternating the role of the General.
2. Alice, Bob, Charlie, and Dave take the majority of the three outputs.

Theorem 2. *The four-party Lamport-Shostak-Pease Protocol is a Byzantine agreement protocol secure against one corrupt party.*

Proof. By Claim 1 each party will observe the same values after round 1, so their outputs in round 2 will be the same. Since at most one of Alice, Bob, and Charlie is dishonest, if all the honest parties' inputs were equal, by Claim 1 two of the three values produced in round 1 will equal this value and so will the output. □

This protocol works fine with three parties only, provided we can solve Byzantine Generals for three parties (one General and two Lieutenants). Suppose the General announces his proposal to the Lieutenants. If the Lieutenants forward their announcements to one another, the honest Lieutenant may be getting conflicting information from the other two parties. This problem can be avoided if the General's proposal can be authenticated by a digital or physical signature.

Authenticated three-party Byzantine Generals:

1. The General announces his proposal to the Lieutenants and signs his message.
2. The Lieutenants forward the General's signed announcement to one another.
3. A Lieutenant receives two signed announcements. If the values and signatures match, he outputs the value. If only one signature matches the General's, he outputs that value. Otherwise he outputs a default value, say zero.

If the General is honest the two signed messages he sends out are equal. Then the cheating Lieutenant cannot prevent the honest one from outputting the General's proposal without forging the General's signature, so the protocol is correct assuming signatures are unforgeable.

If both Lieutenants are honest then they receive the same two announcements in different order so they output the same value.

If the purpose of Byzantine Agreement is to implement a public-key infrastructure then the assumption that signatures can be authenticated may not be realistic. It turns out that this assumption is necessary.

Theorem 3. *Unauthenticated 3-party Byzantine Agreement with one corrupt party is impossible.*

Proof. Suppose P is a three-party protocol with parties Alice, Bob, and Charlie. Assuming P solves Byzantine Agreement, we will construct a six-party protocol P' and then show that in this P' one of the parties must output both 0 and 1 at the same time. This is impossible so the three-party Byzantine agreement protocol P cannot exist.

The protocol P' has six parties A, B, C, A', B', and C'. In P', both A and A' play the role of Alice in P, and similarly for Bob and Charlie. Each party in P' simulates the corresponding party in P with the following communication template. When Alice sends a message to Bob in P, A sends a message to B in P'. But when Alice sends a message to Charlie in P, A sends a message to C' in P'. The complete communication template of P' is depicted in Figure 1.

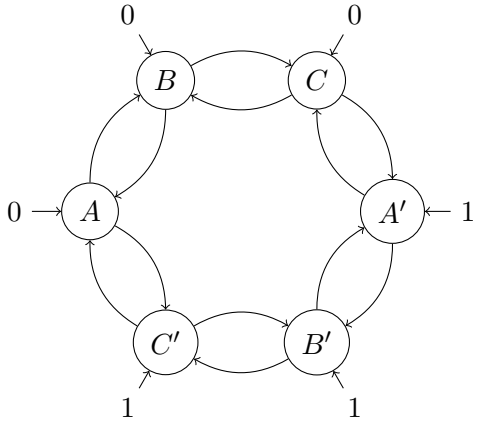

Figure 1: The communication template and the inputs in protocol P'.

Consider an execution of P' in which all parties are honest. The joint view of parties A and B in P' is identical to the joint view of honest Alice and Bob in P when interacting with the following corrupt Charlie. Corrupt Charlie simulates parties C, A', B', and C', delegating Alice's inputs and outputs to C' and Bob's inputs and outputs to C. Since Alice and Bob are honest and P is a Byzantine Agreement protocol, the inputs and outputs of A and B in P' must satisfy the correctness conditions for Byzantine Agreement for honest parties: A and B must output the same value, and this value must match their inputs if they are the same.

By symmetry the same argument applies to all pairs of adjacent parties in P', so we conclude that everyone must output the same value. Now suppose the parties' inputs are like in Figure 1. Then A and B must both output 0 while A' and B' must both output 1. This is impossible. □

Corollary 4. *Unauthenticated $3n$-party Byzantine Agreement with n corrupt parties is impossible.*

Proof. Suppose it was possible. Consider a three-party protocol in which Alice, Bob and Charlie are represented by the first, middle, and last n parties. Alice, Bob, and Charlie give the input to their representatives, run the $3n$-party protocol, and produce any one of their outputs as their own. Without loss of generality assume that Alice is corrupt. Then the $3n$-party protocol still reaches agreement even when all of Alice's representatives are corrupt, so Bob's and Charlie's representatives outputs are equal and equal to Bob's and Charlie's inputs if possible. Therefore Bob and Charlie have reached agreement in the three-party protocol. The three-party protocol is then a Byzantine agreement protocol, contradicting Theorem 3. □

2 Some Byzantine Agreement protocols

Owing to Corollary 4, we can only hope for a protocol under the assumption that a supermajority (strictly more than two thirds) of the parties is honest. Several such protocols have been proposed but none of them is perfect. Some are relatively simple but very inefficient. Others are fairly complicated and still require many rounds of communication. Yet others make additional assumptions on the infrastructure.

An example of the third type is this "magic coin model" protocol by Micali. The infrastructure requirement is that before any given round of the protocol the parties receive a common random bit B (the magic coin). It is assumed that this bit is independent of the previous rounds of the protocol. To illustrate it we assume that there are 7 parties out of which 5 are honest and 2 are corrupt.

Micali's protocol: Repeat the following r times. Initially, each party's value v is its input bit.

0. The parties receive the random bit B.
1. Each party announces its value to the other parties.
2. If a party sees 5 or more announcements for value v, it (re)sets its value to v.
   If not, it sets its value to B.

At the end each party outputs its value v.

Theorem 5. *Micali's protocol reaches Byzantine agreement in the magic coin model except with probability at most* 2^{-r}.

Proof. If all honest parties hold the input v at the beginning of a round, they will hold it at the end of the round. So once all parties agree on a value, they are guaranteed to output this value.

It remains to show that they fail to agree on a value with probability at most 2^{-r}. We show that the failure probability in each round is at most half and these probabilities are independent. Let m be the majority bit among the honest parties' inputs. Then at most 2 of the honest parties' inputs can equal the other value $1 - m$. As there are 2 cheaters, no party can see 5 announcements for $\overline{m}$ in round 2. So each party sets its value to either the honest majority m or the magic coin B. Since m and B are equal with probability half the parties agree on m with probability at least half. □

One of the motivating applications for Byzantine agreement is the implementation of a replicated state machine. There the values represent the state of a system and the parties are distributed servers running local copies of the system. Owing to failure some of the state copies may get corrupted, which can trigger arbitrary behavior from the corresponding party. Byzantine agreement

enables the honest parties to agree on a common state. In this type of scenario the space of values is much larger than a single bit.

Turpin and Coan designed a clever protocol that reduces the problem of agreeing on a value from some very large set like the state of a system (or a hash of it) to agreeing on a single bit. We again illustrate it in the 7-party case with 5 honest parties. This is the first round:

1. Each party announces its input value, which now comes from a large set of possible values. If a party sees at least 5 announcements for v, it (re)sets its value to v; otherwise it resets it to the special value `null`.

Just like in Micali's protocol, this transformation preserves the agreement between the honest parties. Moreover, even if the honest parties disagree, there is at most one choice of non-`null` value v that the honest parties may take at the end of the first round.

Claim 6. *If a party sets its value to some non-*`null` *v then v must be the majority value among the honest players' inputs.*

Proof. The party must have seen at least 5 vs. Since there are at 2 corrupt parties at least 3 of the honest parties' inputs must equal v. □

We are now in a position where each party's value is either v or `null` so the parties can reach agreement via the bit-valued protocol. Unfortunately there is a minor annoyance. Say the parties represent v by the bit 1 and `null` by the bit 0 in the execution of the bit-valued protocol. Suppose they agree on 1. So each honest party is supposed to output v. But some honest parties might have never seen the value v! To rule out this possibility the protocol is augmented with an extra round in which the parties communicate the value v.

Turpin-Coan protocol: Initially, each party's value v is its input bit.

1. Each party announces its (input) value. If a party sees at least 5 announcements for v, it (re)sets its value to v; otherwise it resets it to the special value `null`.
2. The parties repeat step 1.
3. The parties run the bit agreement protocol with the bits 0 and 1 representing `null` and non-`null` values.
4. If bit agreement outputs 0 the corresponding party outputs `null`. Otherwise it outputs the plurality value (the most likely value) it has seen in round 2.

Theorem 7. *The Turpin-Coan protocol is a Byzantine Agreement protocol for 7 parties out of which 5 are honest.*

Proof. If all five honest parties start with the same value v as their input then this value is preserved in steps 1 and 2, all will agree on 1 in step 3, and all will output v in step 4.

Otherwise, by Claim 6 all the honest parties' values after round 1 are the honest majority m or `null`. In either case, the plurality value in round 2 must be either m or `null`. Since step 3 reaches bit agreement the honest parties will all output either m or `null` as desired. □

3 Blockchains

In permissionless distributed systems like the internet it may be unrealistic to assume that a (super)majority of the users is honest. For example recommendation systems are prone to sybil attacks in which a corrupt user can wield disproportionate influence by creating and controlling several identities. Defenses usually rely on a central authority like a government or a mobile provider so that users are authenticated by their ID numbers or mobile phone numbers.

Blockchains are a class of distributed mechanisms that apply incentives to achieve agreement. The information to be agreed upon represents a set of records such as financial transactions. The records are organized into fixed-size blocks (say each block is 1000 records). The blocks are ordered in a linear sequence called the blockchain. The purpose of the mechanism is to ensure that honest users agree on the same blockchain. There are three components:

1. **Record validation:** The validity of the sequence of records in the blockchain can be efficiently certified (by any user). For example, suppose that users are represented by their public keys and records represent financial transactions between them. A record might have the form

   $$Sign(SK_{\text{Alice}}, \text{“3. } PK_{\text{Alice}} \text{ transfers \$1 to } PK_{\text{Bob}}\text{”})$$

   with the intended meaning that Alice wants to transfer \$1 to Bob and this is Alice's third transaction. A user can then authenticate the blockchain records and verify that they are consistent with financial requirements (say Alice's balance can never go negative).

2. **Block-pair validation:** There is an efficient validation function V that takes as inputs a pair of candidate blocks a and b and outputs 1 if both blocks are valid and block b is the successor of block a in the chain.

3. **Tie-breaking:** In case two candidate blockchains satisfy both conditions, the longer one is validated.

How does anyone get any cash to begin with? How is the function V implemented? What happens if two chains are equally long? And how do users end up reaching agreement? The answers to these questions are dictated by the incentive structure of the blockchain mechanism, which is based on the following two principles:

1. Given the last block a in the chain, the task of extending the chain by a block b so that the pair (a, b) passes block-pair validation is relatively difficult. Users seeking to accomplish this task are called *miners*.

2. If a user finds such a block, he is rewarded by cash. The cash reward is represented by adding a special record of the form

   $$\text{“}PK_{\text{Mike}} \text{ mines } x \text{ dollars.”}$$

   Each block has only one record of this type. The reward x is determined by system design. For example in Bitcoin rewards decrease at an exponential rate: Mining can conceivably go on forever, but the total amount that can be mined is finite.

There are different proposals about how to implement V. In a *proof of work* the miners are expected to expend a lot of computational effort. Here is a candidate implementation in the random oracle model. Each block b has a special register that the miner can set to any value. Validation passes if $R(a, b)$ is a string that starts with 50 zeros, where R is the random oracle. On average, a miner has to make 2^{50} calls to R to find a valid setting for the special register.

As the mining task is very consuming (but not impossible), we may expect that one of the miners emerges as the winner some time ahead of the others. He then rushes to announce his solution b to the users in order to claim the mining reward. The tie-breaking requirement motivates the other miners to quickly add b to the tail of the blockchain and work at extending it further, thereby ensuring agreement.

On the other hand, miners have a competing incentive to reject a fellow miner's solution. Suppose Alice transferred \$1 to Bob in the last block and received a cookie in return. She is then motivated to invalidate the last block and pretend that the transfer never took place. If Alice controls more than half of the miners she can make this happen by eventually providing a longer competing extension which will become the agreed-upon blockchain. *Proofs of stake* are alternative mechanisms that address this apparent weakness but that's a topic for another time.