Lecture 3: Proof of Work and Nakamoto Consensus

Principles of Blockchains, University of Illinois,
Professor: Pramod Viswanath
Scribes: Suryanarayana Sankagiri and Xuechao Wang

February 2, 2021

Abstract

This lecture covers the Nakamoto consensus protocol. Coupled with the blockchain data structure, the protocol realizes a decentralized ledger, allowing multiple parties to write to it in a consistent manner. Moreover, the protocol is such that it is robust against an adversary that tries disrupt it; the honest parties continue to have a consistent, ever-growing ledger. In the context of this protocol, we introduce the terms mining, Proof of Work, longest-chain rule and k-deep rule. We also discuss what it means for such a decentralized ledger to be secure.

In the last lecture, we saw that the blockchain data structure enables a tamper-evident and tamper-resistant ledger but with a caveat: only a single party has the privilege to write into the ledger. Other parties may merely read the ledger and verify whether or not it is consistent. To create a decentralized ledger, we showed that the following three questions must be answered.

1. Who are the set of users that can participate in and how are they chosen?
2. When and which block does a user get to append and how do others verify this rule in a decentralized manner?
3. Where does a user append the block? In principle, a block can be appended to any other block in the view of the user.

In this lecture we see how Bitcoin (and many other cryptocurrencies that followed) answers these questions via the **Nakamoto consensus protocol**.

Intuition behind Nakamoto consensus

The Nakamoto consensus protocol, first introduced in the Bitcoin whitepaper, can be described briefly as follows. At any given time, there are a certain number of users that are actively participating in writing to the decentralized ledger, one block at a time. These users can change with time; new users can join in this task of their own free will, without anyone's permission. At regular intervals, a user is chosen at random among the ones currently present to propose the next block. This user creates a block with new data, and the hash pointer of the last block in the blockchain. In effect, it appends the new block to the end of the blockchain, thereby extending the ledger. It then signs this block and broadcasts it to all other users in the system. Other users receive this block, perform some checks on it, and adopt it into (their local copy of) the ledger. The process repeats indefinitely. The process is initialized from a **genesis block**, which is known to all users right from the beginning.

How can such a system be realized? In particular, how can one randomly pick a user to propose the next block, especially when the number of users is variable? Nakamoto's idea was to use a simple

concept called *proof-of-work*, described next. This concept is a salient feature of many blockchain systems, and arguably the most important novelty of Bitcoin. It provides an answer to the first two questions above.

What other aspects go into the creation of a new block? Firstly, the miner must include new data-values that have not been included in the ledger so far. This aspect of the protocol will be covered in the coming lectures. Secondly, it must include the hash pointer of a block that is already on the ledger: it should the new block should point to the last block currently on the ledger. Formally called the *longest-chain rule* for block proposal, this is another salient feature of the Nakamoto consensus protocol. It answers the third question above. This seemingly innocuous rule has important implications for the security of the protocol, which become clear when we consider adversarial behavior, i.e., users who do not follow this rule.

Proof-of-work and mining

Recall that hash puzzles are a game in which one tries to find a nonce (an integer) such that H(nonce, data) is less than some threshold. Here, data is some pertinent string; every new piece of data defines a new hash puzzle. If the the threshold is small, one needs to try out many different nonces in order to solve the puzzle. For each nonce, one most re-compute the hash function. Every hash computation takes some non-zero time. The exact time depends on the length of the data string; for the sake of estimates, we can assume it to be a few nanoseconds on a modern computer. A party that has solved such a hash puzzle must clearly have tried a large number of nonces and computed hashes with each one of them. Thus, it must have put in a lot of 'work' , i.e., computation time. It simply cannot guess a nonce that solves the puzzle in a short time (except with small probability, purely by luck). We say that a nonce that solves the hash puzzle (in conjunction with the pertinent data) serves as a '**proof-of-work**' (PoW for short).

Now imagine multiple parties competing to solve the same hash puzzle the earliest. As we saw, there is no real strategy; they must simply try random nonces and re-compute the hash each time with a different nonce. Since multiple parties are working towards the same puzzle, it will get solved faster. This is because the search for different nonces is now split among different computers which work in parallel. Which party solves the puzzle the first? A priori, this cannot be determined; the winner of this race is random, beyond the control of the competing parties. At best, a party can improve its chances by buying a better computer which computes hashes faster. Even so, it is unlikely that one party has much more computing power than all the others combined. Thus, multiple parties have a reasonable chance of winning the race. Moreover, the winner changes from one hash puzzle to another.

The process of searching for a nonce that solves the hash puzzle is called **mining**. The precious resource being mined here is the right nonce, i.e., the proof-of-work. Just as one must keep digging (mining) until one strikes gold, one must keep computing hashes with random nonces until one solves the puzzle. The parties competing to solve the hash puzzle are called **miners**. In the Nakamoto consensus protocol, what is the gold that the miners get upon solving the hash puzzle? In a nutshell, they get to propose a new block. Before we specify the details, let us note that such a system has the properties described in the previous section, i.e., new blocks are proposed at some regular intervals of time (with some random fluctuations), and that the proposer is randomly chosen among the currently active users.

We now elaborate on how hash puzzles are used in the context of blockchains. In blockchains operating with the Nakamoto consensus protocol, each block contains a nonce (this is simply a special field in the block). The **block header** consists of the Merkle root of data in the block, the hash pointer to the previous block, the nonce, the timestamp of the block, and perhaps some other meta-data relevant to the application. A block is considered *valid* only if the hash of its block header is less than some pre-specified threshold (more on how this threshold is chosen later). In Bitcoin,

the threshold is set such that a new block is expected every ten minutes. Note that the hashes are computed only on the block header and not on the whole block data, which makes it more efficient to compute them.

At any given time, each miner m creates a block B_m with some data, and includes the hash pointer of the latest known block (i.e., the block at the end of the ledger) into B_m. For different miners m, B_m may contain identical data, or may differ. All miners search for nonces to solve the hash puzzle. The first successful miner immediately broadcasts its block, with the proof-of-work nonce to other users. We say that a *new block is mined*; let this block be B. These other users first check the proof-of-work (simply check whether the new block's header that they receive has a small enough hash). If this criterion is satisfied, they then create a new block, B'_m, locally, with fresh data and a hash pointer pointing to B. We say that the miners mine on top of B. The process then repeats. This completes the description of the mining process in a proof-of-work system.

Mining as a Poisson process

In this section, we model the mining process mathematically. This mathematical model gives a good intuition of the process, and serves as a basis for theoretical calculations later on. The times at which a new block are mined is modeled as a Poisson process with rate λ. A Poisson process is one in which new events (or arrivals) occur at random intervals with intervals exponential distribution. Moreover, the intervals between any two events are independent of each other. Recall that an exponential random variable X with parameter λ has distribution

$$\mathbb{P}(X \geq t) = \exp(-\lambda t) \ \forall t \geq 0.$$

Also recall that a Poisson random variable Y with parameter λ has distribution

$$\mathbb{P}(Y = n) = \exp(-\lambda)\frac{\lambda^k}{k!} \ \forall n \geq 0.$$

In a Poisson process, the number of events in an interval of length T is a Poisson random variable with parameter λT. Moreover, the number of events in disjoint intervals of time are independent. If we consider small intervals ($\lambda T \ll 1$), there is one event in the interval with probability λT and none otherwise. Thus, a Poisson process can be emulated by counting the occurrence of heads in a sequence of (independent) coin tosses, with the probability of heads being very small. We now see why the mining process has such a property.

Imagine that the number of miners in the system and the total computing power at their disposal is constant over some period of time. Assume that each of the miners' computers are mining incessantly, and that is the only computation they are performing. It is then reasonable to say that the total number of hashes being computed (equivalently, the total number of nonces being tried out) per unit time is roughly constant. Say, a billion hashes are computed every second.

Further, suppose that the difficulty level of the hash puzzle for proposing a block is very high. E.g., say that the first thirty-five zeros must be zero for a block to be valid. The probability that a particular nonce will meet this criterion is $2^{-35} \approx 3 \times 10^{-11}$. Thus, the probability that the hash puzzle will be solved by *any miner* in a given second is 0.03, a small number (we assumed that a billion hashes are computed every second). Whether or not a proof-of-work hash is found in a particular second has no bearing on whether one will be found in the next second. The reason for this is that the hash function is essentially a random oracle; the hash values for different inputs are independent of each other. Thus, the mining process is well modeled as a Poisson process.

In modeling the mining process as a Poisson process, we focus only at the times at which new, valid blocks are created. The Poisson process model holds irrespective of the number of miners, their individual computation power, whether different miners are working on the same block or different blocks, and when different users receive newly mined blocks. The parameter λ of the mining process, which is equal to the average number of blocks mined per unit time, is called the **mining rate**.

Forks and the longest-chain rule

Note that the mining rate is different from the rate at which the ledger grows. Ideally, each new block should lead to the growth of the ledger. However, this is not always the case. For example, it is possible that a miner mines a valid block but does not publish it; in this case, the ledger does not grow (this is dishonest behavior, but we must account for it nevertheless). It is also possible that two valid blocks are mined with the same parent block, in which case the ledger grows only by one block. This can happen if the second hash puzzle is solved before the pertinent miner heard of the previous block. After all, it takes a non-zero amount of time for a block to be communicated across the network, especially if the block has a lot of data.

In general, the set of blocks mined at any given point in time form a directed tree, rather than a single chain. We say that the blockchain has **forked** when a single block has two or more children blocks. We saw how forks might occur due to communication delays or adversarial behavior in the previous paragraph. As such, there could be many forks over time. What then should the users consider as 'the ledger'? The **longest-chain rule** states that the longest chain among all published blocks should be treated as the ledger. Thus, users should build a new block and append it to the longest chain that they currently know of. When there are ties (i.e., there are two or more chains of equal length, that fork at some level), one favors the branch of the fork where the blocks' hashes are the smallest.

Adversarial users

The description so far describes the salient aspects of the Nakamoto consensus protocol. (Some more details will be covered in the next lecture). All users are expected to follow the protocol exactly. In reality, there may be some who deviate from the protocol; they are **adversarial users** (aka corrupt/malicious users). Those who do follow the protocol are called honest parties. What might be the aim of the adversarial parties? In general, their aim is to disrupt the system in any possible way. Here, we describe one possible attack on the append-only property of the ledger, called the **private attack**.

In the private attack, a group of adversarial users mine new blocks, but keep these blocks private to themselves. Honest users are simply unaware of these private blocks, and continue to mine as if these blocks never existed. In effect, the adversarial users have created a fork in the blockchain, but one that only they can see. Suppose, due to randomness in the mining process, the adversarial users get lucky and build a few (say, five) private blocks in quick succession. During the same interval, say the honest users only mine three blocks. In such a case, the private (adversarial) chain is longer than the public (honest) chain. Now, the adversarial users release their private chain to all other users. By the longest-chain rule, these honest users must give up their chain and adopt the adversarial chain. In effect, the last three blocks have been erased from the ledger.

The k-deep confirmation rule

In the Nakamoto consensus protocol, such changes to the end of the ledger is impossible to avoid. Even if we assume that there are no adversarial users, we have seen that forks can occur because of network delay. Wherever there are forks, there is the possibility that the longest-chain may switch, which implies that the last portion of the ledger is re-written. The solution around this is simple and intuitive. Users should treat a block as **confirmed** only if it is buried deep enough under other blocks. Put differently, only if some k blocks are built directly below a block B, should the block B be confirmed. Confirming a block means that the portion of the ledger up to that block is now immutable. It is not advisable to confirm very recent blocks as they may be over-written, due to forks in the blockchain.

The k**-deep rule**, mentioned above, reflects the belief that any change in the longest-chain are towards the end of the chain. The prefix of the longest chain at some time t, i.e. the chain obtained by dropping the last k blocks, continues to remain a prefix of the longest chain at future times too. Note that for any finite value of k, it is not guaranteed with absolute certainty that this will be true. If the adversarial users invest in a lot of computing power, block communication among honest users, or simply get very lucky, they could even overturn a deeply buried block. However, larger the value of k, the more unlikely it is that this happens (as long as protocol-following miners possess the majority of the total computation – the so-called *hash power*). In fact, as we show below, the probability that this happens decreases exponentially with k.

Intuition for security–the private attack

Consider a simple model of the blockchain system with all users split into two groups, honest and adversarial. The adversarial users are trying to perform a private attack, by overturning the last k blocks in the ledger. Assume that the computing power of each user remains fixed, with the total mining rate being λ. Let the fraction of "hash power" with adversarial users be β, and assume that $\beta < 1/2$. This means that adversarial users mine blocks as a Poisson process of rate $\beta\lambda$, while honest users Further, assume that communication among all parties is instantaneous, i.e., there are no communication delays. This means that after the private attack is launched, the private chain (visible only to adversarial parties) and public chain (visible to all parties) grow at rate $\beta\lambda$ and $(1-\beta)\lambda$ respectively.

With these parameters set, whether or not the private attack will be successful depends only on the randomness of the mining process. We say that the private attack is successful (from block B) if it overturns k or more honest blocks (built below block B). For this to happen, over some long-enough interval, the adversarial users must mine more blocks than honest ones. For simplicity, suppose the private attack is launched right from the beginning, i.e., launched from the genesis block. With $\beta < 1/2$, the adversary must be lucky (mine blocks quicker than expected), or the honest users must be unlucky (mine blocks slower than expected), or both. If not, the attack cannot succeed.

The number of honest blocks in the first T units of time, X_T, is random with distribution $\mathsf{Poisson}((1-\beta)\lambda T)$, which has mean $(1-\beta)\lambda T$. The same holds for adversarial blocks (denoted by Z_T), but with $1-\beta$ replaced by β. The Chernoff bound on Poisson random variables states that if $X \sim \mathsf{Poisson}(\lambda)$, then

$$\mathbb{P}(X \geq \lambda + x) \leq \exp(-x^2/2(\lambda + x)) \tag{1}$$

$$\mathbb{P}(X \leq \lambda - x) \leq \exp(-x^2/2(\lambda + x)) \tag{2}$$

Let $\epsilon \triangleq 1 - 2\beta$. Using (1) for Z_T with $x = (1/2 - \beta)\lambda T$ gives

$$\mathbb{P}(Z_T \geq 0.5\lambda T) \leq \exp(-\epsilon^2 \lambda T/4)$$

Using (2) for X_T with $x = (1/2 - \beta)\lambda T$ gives

$$\mathbb{P}(X_T \leq 0.5\lambda T) \leq \exp(-\epsilon^2 \lambda T/4(1+2\epsilon)) \leq \exp(-\epsilon^2 \lambda T/12)$$

Thus, the probability that Z_T exceeds X_T for any value of T greater than T_0 is bounded above by (due to the union bound)

$$\begin{aligned}\sum_{T=T_0}^{\infty} \mathbb{P}(Z_T \geq X_T) &\leq \sum_{T=T_0}^{\infty} \mathbb{P}(Z_T \geq 0.5\lambda T) + \mathbb{P}(X_T \leq 0.5\lambda T) \\ &\leq \sum_{T=T_0}^{\infty} \exp(-\epsilon^2 \lambda T/12) + \exp(-\epsilon^2 \lambda T/4) = C(\epsilon, \lambda) \exp(-\epsilon^2 \lambda T_0/12)\end{aligned}$$

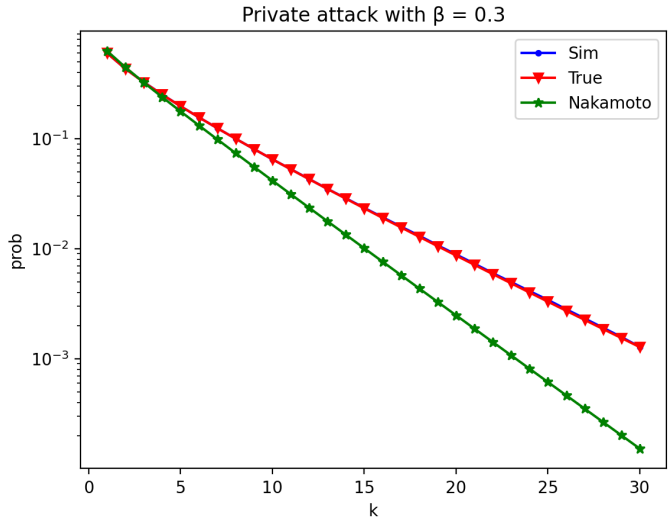

Figure 1: Private attack with $\beta = 0.3$

where $C(\epsilon, \lambda)$ is some constant that depends on ϵ and λ. What value of T_0 should we choose? We should choose a value such that k honest blocks take more than T_0 time to appear. Let T_0 be such that $\lambda T_0 = k$, which implies $(1-\beta)\lambda T_0 < k$. The probability that $X_{T_0} \geq k$ is bounded by $\exp(-\beta^2 \lambda T_0/8)$. Thus, the probability of a private attack with parameter k is bounded by $\exp(-\beta^2\lambda T_0/8)$ $+C(\epsilon,\lambda)\exp(-\epsilon^2\lambda T_0/12)$. replacing λT_0 by k shows that the probability of this happening decays exponentially with k.

However, private attacks are not the only strategy the adversary can employ. We will see other attacks in later lectures. However, one can show that the private attack is actually the worst case attack in term of success probability for the longest chain protocol. The logic is to show that for a fixed sample path in the probability space, if any other attack succeeds, then the private attack also succeeds. We can also calculate the exact success probability of the private attack. Details of the proof and the calculation can be found in the Appendix. Nakamoto [3] himself/herself made a subtle mistake in the calculation, which was first pointed out in [4].

Variable mining difficulty

A key requirement of deployed PoW blockchains is to adapt to the immense variation in mining power. For example, the mining power of Bitcoin increased exponentially by an astonishing factor of 10^{14} during its decade of deployment. This is a reflection of the growing popularity of Bitcoin; many people have invested a lot of computational resources into it over time. If Bitcoin had continued to use the same difficulty for the hash puzzle, then the average inter-block time would have fallen from the original 10 minutes to 6 picoseconds. By adjusting the difficulty threshold of Bitcoin, using a difficulty adjustment algorithm, the average inter-block time is kept constant over the course of a long duration of time.

We call the hash puzzle threshold in PoW mining the *target* of a block. The *difficulty* of each block is measured in terms of how many times the block is harder to obtain than using the initial target of the system that is embedded in the genesis block. The *chain difficulty* of a chain is the sum of difficulties of all blocks that comprise the chain, then each block in the chain *covers* an interval of chain difficulty. We also refer the chain difficulty of a block as the chain difficulty of the chain

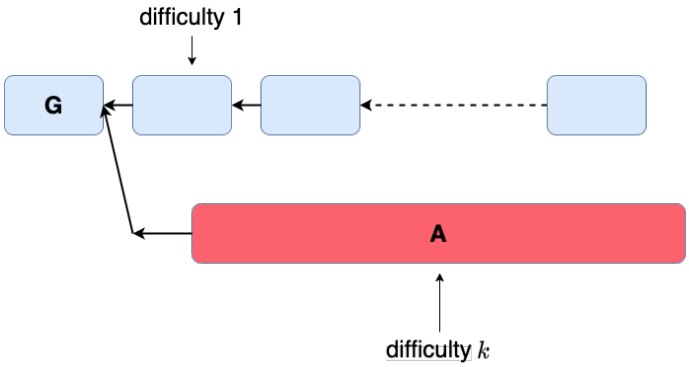

Figure 2: A simple attack if allowing miners to choose their own difficulty. The adversary mines one block which is as difficult as k honest blocks.

ending at this block. There are three core ideas to the Bitcoin difficulty adjustment algorithm: (a) vary the difficulty target of block mining based on the average inter-block time from the previous epoch (of 2016 blocks), (b) use the *heaviest* chain (calculated by the sum of the block difficulties) instead of the longest chain to determine the ledger, and (c) allow the difficulty to be adjusted only mildly every epoch (by an upper bound of a factor of 4). While this appears to be a simple and intuitive algorithm, minor seemingly-innocuous variants turn out to be dangerously insecure.

Consider a simpler algorithm using only (b), i.e., simply let the miners choose their own difficulty and then use (b) the heaviest chain rule. At a first glance, this rule appears kosher - the heaviest chain rule seems to afford no advantage to any miner to manipulate their difficulty. However, this lack of advantage only holds in expectation, and the variance created by extremely difficult adversarial blocks can thwart a confirmation rule that confirms deeply-embedded blocks, no matter how deep, with non-negligible probability proportional to the attacker's mining power. We give a simple calculation here. Suppose honest miners are adopting the initial mining difficulty as defined in the genesis block, with expected inter-block time being 10 minutes. Let 10 minutes be our unit of time and the initial difficulty be the difficulty unit, hence on average it take k units of time to mine a honest chain with k blocks. See Figure 2 for illustration. Suppose the adversarial mining power is half of the honest mining power (or 1/3 of total mining power). To mine a heavier chain, the adversary only needs to mine one block which is as difficult as k honest blocks, within k unit of time. Then the adversarial mining process follows a Poisson point process with rate $1/2k$, and the number of adversarial blocks mined in k unit of time follows the Poisson distribution $\mathsf{Poiss}(1/2)$. Hence the success probability of this attack would be

$$\mathbb{P}(\text{attack succeeds}) = \mathbb{P}(\mathsf{Poiss}(1/2) \geq 1) = 1 - \mathrm{e}^{-1/2} \approx 39.3\%,$$

which is a constant independent of k, therefore any k-deep confirmation rule will fail.

Now consider a more detailed rule involving only (a) and (b). It turns out that there is a difficulty raising attack [1], where the adversary creates an epoch filled with timestamps extremely close-together, so that the difficulty adjustment rule from (a) will set the difficulty extremely high for the next epoch, at which point, the adversary can utilize the high variance of the mining similar to the aforementioned attack. Now we describe this attack in detail. Note that the adversary can put any timestamp in its private blocks, so the difficulty of the second epoch in its private chain can be arbitrary value as long as the adversary completes the first epoch. Let B with difficulty X be the first block of the second epoch in the private chain, then B has chain difficulty $2016 + X$. See Figure 3 for illustration. To mine an honest chain with chain difficulty $2016 + X$, on average it takes $2016 + X$ unit of time. On the other hand, considering the same adversary, it takes on average

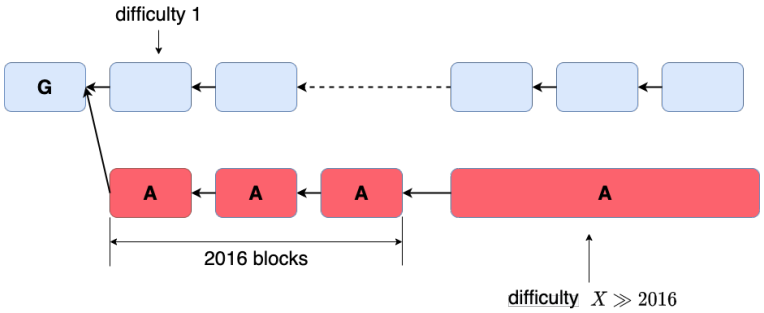

Figure 3: The difficulty rising attack. The adversary raises the difficulty to extremely high in the second epoch by faking timestamps.

4032 unit of time for it to complete the first epoch in its private chain. Therefore, to succeed in this attack, the adversary needs to mine the block B within $X - 2016$ unit of time, which happens with probability:

$$\mathbb{P}(\text{attack succeeds}) = \mathbb{P}(\mathsf{Poiss}(\frac{\mathrm{X} - 2016}{2\mathrm{X}}) \geq 1) = 1 - \mathrm{e}^{-\frac{\mathrm{X}-2016}{2\mathrm{X}}} \approx 1 - \mathrm{e}^{-1/2} \approx 39.3\%,$$

if $X \gg 2016$. Note that the success probability is independent of the length of the public longest chain, hence any k-deep confirmation rule will fail again.

This more complex attack is only thwarted using the full protocol that employs (a), (b) and (c) together. Formally, the Bitcoin difficulty adjustment algorithm is as follows: Consider a chain of v blocks with timestamps $(r_1 \dots r_v)$. For fixed parameters τ (= 4 in Bitcoin), Φ the length of an epoch in number of blocks (= 2016 in Bitcoin), Λ_0 the expected duration of an epoch (= 2 weeks in Bitcoin). The target calculation function $D : \mathbb{Z}^* \to \mathbb{R}$ is defined as

$$D(\epsilon) = T_0,$$
$$D(r_1 \dots r_v) = \begin{cases} \frac{1}{\tau}T & \text{if } \frac{\Lambda}{\Lambda_0}T < \frac{1}{\tau}T \\ \tau T & \text{if } \frac{\Lambda}{\Lambda_0}T > \tau T \\ \frac{\Lambda}{\Lambda_0}T & \text{otherwise} \end{cases}$$

where T_0 is the initial target as defined in the genesis block, and Φ', Λ, and T correspond to the last block, duration, and target of the last completed epoch, respectively, i.e., $\Phi' = \Phi\lfloor v/\Phi \rfloor$, $\Lambda = r_{\Phi'} - r_{\Phi'-\Phi}$ and $T = D(r_1 \dots r_{\Phi'-1})$. A full and beautiful analysis of Bitcoin rule is provided in [2].

Bitcoin is Permissionless

In Bitcoin, participants can generate a new (secret key, public key) pair for themselves at any point in time. Thus, a single user can pretend to be multiple different people. In fact, doing so is encouraged for privacy reasons. Such a system is called a **permissionless system**. However, creating multiple identities in Bitcoin does not truly increase one's representation in the system; that is determined by the mining (computation) power, which can only be grown by a capital investment. If the system were such that an adversary could gain advantage by creating multiple identities, then such an attack is called a **Sybil attack**. Bitcoin inherit the **Sybil-resistance** property from PoW mining. Thus the PoW mining process simultaneously achieves multiple goals in Bitcoin: (a) Sybil-resistance; (b) Randomized block proposer election; (c) Adjusting the average inter-block duration time. In some other blockchain designs, there are external mechanisms to ensure that each entity only has a single

key. Such a system is said to be *permissioned*. These blockchains will also have separate mechanisms for items (b) and (c). We will see some of these designs in later lectures.

References

[1] Lear Bahack. Theoretical bitcoin attacks with less than half of the computational power (draft). *arXiv preprint arXiv:1312.7013*, 2013.

[2] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. Full analysis of nakamoto consensus in bounded-delay networks. Cryptology ePrint Archive, Report 2020/277, 2020. https://eprint.iacr.org/2020/277.

[3] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[4] Meni Rosenfeld. Analysis of hashrate-based double spending. *arXiv preprint arXiv:1402.2009*, 2014.

Appendix

In this appendix, we prove that the private attack is one of the optimal adversary strategies in terms of probability of success when the adversary tries to create a fork from the **genesis** in the zero network delay case (this is an idealization of the Bitcon parameter setting of mining rate of one block very ten minutes, which is much much smaller than network propagation delay).

Optimality

Security model: There is zero delay among honest nodes so **honest blocks always appear on different heights**. If there is more than one longest chain, then the adversary controls how the honest party's mining power is split across the multiple longest chains. We assume the adversary launches the attack from the genesis block and starts mining at the same time as honest nodes (i.e., no pre-mining phase). We say the attack succeeds when the adversary at some time creates a fork from the genesis and both chains are of equal length at least k. In Nakamoto's private attack, the adversary simply mines a chain from the genesis privately and releases the private chain when the private chain exceeds the honest chain and the honest chain has length at least k.

Proof. We use a random 0-1 string $w^{(n)} \in \{0,1\}^n$ to represent the randomness in the attack. $w_i^{(n)} = 0$ means that the honest nodes mine the i-th block and the honest block will be placed to one of the two chains according to the longest chain rule. If there is a tie, tie breaking can be in favor of the adversary. $w_i^{(n)} = 1$ means that the adversary mines the i-th block and the adversary can take arbitrary action (eg., keep it private, publish it, or even drop it). For any fixed k, given an adversarial strategy S, if the adversary can create a fork with length at least k under string $w^{(n)}$, we call $w^{(n)}$ a S-bad string. Note that if S is a randomized strategy, then we consider the worst case. Let $q_n^S = P(w^{(n)}$ is a S-bad string). Let S^* be the private attack. We will prove $q_n^S \le q_n^{S^*}$ for any S.

$$\begin{aligned} q_n^S =& P(w^{(n)} \text{ is a } S\text{-bad string}) \\ \le& P(\exists k \le m \le n/2, \text{such that } \# \text{ of 0's} \le \# \text{ of 1's in the first } 2m \text{ bits in } w^{(n)}) \\ =& P(w^{(n)} \text{ is a } S^*\text{-bad string}) = q_n^{S^*}. \end{aligned} \tag{3}$$

Note that for any strategy S, if $w^{(n)}$ a S-bad string, then both $\overline{w^{(n)}0}$ and $\overline{w^{(n)}1}$ are S-bad strings. So we have $q_n^S \le q_{n+1}^S$, then by Monotone Convergence Theorem, $\lim q_n^S$ exists and we write it as q^S, which is the success probability of strategy S. Applying limit on (3), we get $q^S \le q^{S^*}$ for any S, which concludes the proof.

□

Calculation of q^{S^*}

Let β be the fraction of adversarial mining power. We will compute q^{S^*} as a function of k and β.

Nakamoto [3] also calculates the success probability of private attack. However, Nakamoto assumes that when the honest chain has length k, the length of the private chain will follow a Poisson distribution with expected value $k\beta/(1-\beta)$, which is not correct. Let Z be the length of the private chain at the time when the length of the honest chain reaches k, then the distribution of Z should be

$$P(Z=m) = \binom{k-1+m}{m}(1-\beta)^k\beta^m, \tag{4}$$

for $m = 0, 1, 2, \cdots$. One can check that $E[Z] = k\beta/(1-\beta)$.

From random walk theory, the probability that the private chain can catch up the honest chain from s block behind is $(\frac{\beta}{1-\beta})^s$ when $\beta < 1/2$. So we have

$$\begin{aligned} q^{S^*} &= \sum_{m=0}^{k}\binom{k-1+m}{m}(1-\beta)^k\beta^m(\frac{\beta}{1-\beta})^{k-m} + \sum_{m=k+1}^{\infty}\binom{k-1+m}{m}(1-\beta)^k\beta^m \\ &= 1 - \sum_{m=0}^{k}\binom{k-1+m}{m}[(1-\beta)^k\beta^m - (1-\beta)^m\beta^k]. \end{aligned} \tag{5}$$