A Comparative Analysis of Proof of Work and Proof of Stake Under Sybil, Long-Range, and Selfish Mining Attacks

Final Project Report

BSc Computer Science (Artificial Intelligence)

Author: Nidhushkar Raveelackshman

Supervisor: Christopher Hampson

Student ID: 21085103

April 17, 2025

Abstract

Blockchain systems require robust consensus systems to achieve security and decentralisation. This dissertation conducts a comparative study of Proof of Work (PoW) and Proof of Stake (PoS), representing the two main blockchain consensus methods. This research examines the resistance capabilities of PoW and PoS against three primary attack types: Sybil attacks, long-range attacks, and selfish mining attacks.

This research evaluates the security assumptions behind PoW and PoS through an analysis of attack viability and impact on each protocol, as well as an assessment of mitigation strategies found in previous studies. The analysis reveals that PoS and PoW prevent Sybil attacks through resource-based voting power, but each system has its own distinct vulnerabilities. PoW systems become vulnerable to selfish mining attacks when specific conditions exist, while PoS systems remain exposed to long-range attacks unless they implement additional security measures.

To analyse these vulnerabilities in practice, Python simulations were implemented to model basic blockchain systems during Sybil, long-range, and selfish mining attacks. The simulations used probabilistic analyses combined with economic threshold calculations to determine attack feasibility conditions. The simulation results demonstrate the inherent trade-offs between PoW and PoS in terms of security and decentralisation. This underscores the importance of careful protocol design to guard against these vulnerabilities.

Originality Avowal

I verify that I am the sole author of this report, except where explicitly stated to the contrary. I grant the right to King's College London to make paper and electronic copies of the submitted work for purposes of marking, plagiarism detection and archival, and to upload a copy of the work to Turnitin or another trusted plagiarism detection service. I confirm this report does not exceed 25,000 words.

Nidhushkar Raveelackshman

April 17, 2025

Acknowledgements

I would like to thank my supervisor, Christopher Hampson, for his guidance and support during this project. I am also very grateful to my family and friends for their moral support, patience and encouragement during my studies.

Contents

Chapter 1

Introduction

Blockchain technology serves as the foundation for cryptocurrencies and other decentralised systems by ensuring agreement (consensus) on the state of a distributed ledger without a central authority. Essentially, a blockchain is a distributed ledger (database) composed of a chain of blocks of transactions that are shared and validated by many nodes. All participants reach an agreement on new block contents through cryptographic techniques and a consensus protocol that enables trustless agreement about the ledger state. The two main consensus systems currently in use are Proof of Work (PoW), which Bitcoin implemented in 2009, and Proof of Stake (PoS), which emerged in later developed cryptocurrencies, starting with Peercoin in 2012, as an energy-saving alternative [24]. Both mechanisms are intended to prevent adversarial control of the network by making voting power a function of limited resources. PoW relies on computational effort (hash power), whereas PoS relies on economic stake (cryptocurrency holdings). Effectively, this requirement has a cost associated with joining in consensus, serving as a disincentive to Sybil attacks (where the attacker creates multiple pseudonyms to vote out honest nodes) [19]. By making consensus influence proportional to resources (CPU power or coin stake) rather than to mere identities, PoW and PoS force a Sybil attacker to acquire a majority of the resource, which is presumed difficult and expensive [27]. This design has so far kept major cryptocurrencies decentralised and secure in practice.

However, PoW and PoS systems present unique vulnerabilities which create security issues. In PoW systems, attackers who possess significant hash power can perform 51% majority attacks to reverse confirmed transactions and perform double-spending (the attacker can spend coins

twice). Furthermore, attackers could use more sophisticated methods, including selfish mining, privately forking the blockchain to obtain an unfair share of mining rewards [20].

The transition from PoW to PoS removes the high energy costs. However, it creates new security risks primarily through long-range attacks, also known as history revision or the "nothing-at-stake" problem. An adversary can rewrite the blockchain's history at a low cost by using old private keys through costless forking [29]. Understanding these vulnerabilities becomes essential because the industry currently conducts major network transitions (Ethereum made its PoS shift from PoW in 2022), and attackers evolve their methods.

1.1 Aims and Objectives

This project aims to formally analyse and compare the security of Proof of Work and Proof of Stake blockchains concerning Sybil attacks, long-range attacks, and selfish mining, using both literature study and practical modelling. This aim is approached by implementing a Python-based simulation framework to model simplified PoW and PoS consensus environments under attack scenarios. The specific objectives are to:

- Review the literature on PoW and PoS security, distilling how each consensus mechanism works in practice (e.g., in Bitcoin and Ethereum) and summarising known vulnerabilities or past attacks identified by researchers. This includes identifying the conditions under which Sybil, long-range, and selfish mining attacks become feasible.

- Design and implement simulations for PoW and PoS networks in Python. For PoW, the simulation will model a network of miners racing to find blocks via hashing. For PoS, it will model a set of validators selected to create blocks based on stake. The simulator will allow toggling an "attacker" entity that can carry out selfish mining in PoW or attempt chain forks in PoS.

- Simulate attack scenarios for each attack vector:

    1. **Sybil attack simulation**: Create many bogus nodes and observe if they can influence consensus (e.g., in an unprotected setting) versus a properly weighted setting (PoW hash power or PoS stake).

    2. **Long-range attack simulation**: In the PoS model, allow an attacker with historical stake keys to introduce an alternative history and examine if honest nodes ever

adopt it.

3. **Selfish mining simulation**: In the PoW model, run a selfish miner strategy (with configurable hash power α) against honest miners and measure the selfish miner's success rate and reward share.

- Collect and analyse data from the simulations to compare outcomes. The most important measures include the success probability of the attacker (e.g. probability of a long-range fork overtaking the honest chain, or proportion of blocks acquired by a selfish miner) versus different attack power or stake.

- Evaluate and compare the resilience of PoW vs PoS: Using the simulation results and theoretical expectations, determine which mechanism is more vulnerable to each type of attack and why. For instance, measure the minimum resources needed for an attack to become viable in each system and note any key thresholds (such as the known 30% hash power threshold for profitable selfish mining in PoW [28]).

- List potential mitigations or design choices: According to the findings, discuss how each consensus can be made secure from such attacks. This may include referencing techniques like PoS checkpointing (to prevent long-range attacks) or protocol adjustments proposed in the literature (e.g., uniform tie-breaking rules to mitigate selfish mining).

Chapter 2

Background

2.1 Overview of PoW and PoS Consensus Mechanisms

2.1.1 Proof of Work (PoW)

In a PoW blockchain, network participants known as miners compete against each other to solve cryptographic puzzles. The miner who discovers a valid solution first earns the right to add the following block to the chain while receiving a reward [27]. The probability of mining a block is directly proportional to a miner's computational power (hash rate) relative to the combined hashing power of the entire network [27]. This mechanism naturally limits Sybil attacks: creating many identities (nodes) gives no advantage without proportional computing power. An attacker controlling a fraction q of the total hash power will, on average, mine fraction q of the blocks. For the system to remain secure, honest miners are assumed to hold $> 50\%$ of the total hash power so that no single entity can consistently override the majority decision. If an attacker's hash power is less than the honest majority's, the probability that the attacker can catch up and surpass the honest chain diminishes exponentially as more blocks are added [27]. For instance, Nakamoto's analysis shows that if an attacker with fraction $q < 0.5$ of hash power falls z blocks behind the honest chain, the probability that the attacker ever catches up is given by a decaying tail of a negative binomial distribution:

$$P_{\text{catch-up}} = \sum_{k=0}^{\infty} \binom{k+z-1}{k} q^k (1-q)^z \tag{2.1}$$

which becomes negligibly small as z grows (for any $q < 0.5$) [27]. In simpler terms, each additional confirmation block makes it exponentially harder for a minority attacker to reorganise the chain. This principle supports the current blockchain security protocol that requires multiple confirmations (such as six blocks in Bitcoin) before accepting a transaction as fully validated. An attacker significantly below 50% hash power has essentially no chance to reverse a transaction once it is buried deep in several blocks.

Since mining in PoW consumes substantial real-world resources (electricity and hard- ware), an attacker trying a Sybil or majority attack faces tremendous economic costs [27]. A 51% attack in PoW – where an attacker controls $> 50\%$ of the mining power is theoretically possible but practically infeasible on large networks due to the required hardware investment and energy expense. As of writing, Bitcoin has never been successfully double-spent via a 51% attack, reflecting the deterrence provided by PoW's cost structure. Smaller PoW networks, however, have experienced such attacks when a malicious miner temporarily rented or acquired a majority of hashing power, verifying the assumption that PoW's Sybil resistance is only valid as long as no adversary can afford to gain majority resources.

2.1.2 Proof of Stake (PoS)

PoS emerged to address some of PoW's limitations, specifically its high energy usage. The system distributes block production rights to validators in proportion to the amount of cryptocurrency they own (their stake) rather than their computational work. Validators lock up a portion of their coins (their stake) and are randomly chosen to create new blocks, with the probability being proportional to their stake percentage [24]. For example, a validator who controls 10% of all staked coins has the statistical probability to propose the following block at approximately 10%. The stake-weighted selection method functions similarly to PoW's hash-based competition because it links power to a resource (coins) that is expensive to obtain, thus making attacks economically costly [24]. In principle, a PoS network is also Sybil-resistant – creating multiple validator identities does not increase an attacker's influence unless they also split or increase their total stake. One entity with 1000 coins has the same total forging power, whether those coins back one identity or ten identities of 100 coins each; either way, their overall chance of producing the next block remains 1000/total stake. Thus, like PoW, PoS requires an attacker to accumulate a majority of the resource (stake) to dominate the consensus process, which should be prohibitively expensive in a well-distributed system.

Despite these similarities, PoS fundamentally differs in how consensus is reached, and finality (the irreversibility of blocks) is achieved. Many PoS implementations use additional randomness and committee selection to choose block proposers and validators, often with supplementary protocols to finalise blocks. For instance, Ouroboros (the PoS protocol of Cardano) uses randomised leader election [23], Algorand employs Byzantine Fault Tolerant committees, and Ethereum's Casper finality gadget overlays PoS with a voting mechanism to finalise checkpoints [15]. The security assumption for PoS is typically that honest participants hold a majority of the staked value (e.g. $> 50\%$ or sometimes a supermajority $> 66\%$ depending on the protocol's design) [28]. If an attacker obtains a majority of the stake, they could, in theory, validate a fraudulent chain. However, acquiring over half of all coins is not only extremely costly but may also be detectable and countered (through community intervention or protocol governance). The strength of PoS lies in economic incentives: honest behaviour is rewarded (through block rewards or transaction fees), while malicious behaviour can be penalised (e.g., via slashing, where a portion of a dishonest validator's stake is forfeited for protocol violations) [29]. These incentives and penalties are intended to align participants with the protocol rules.

It is important to note that PoS introduces the concept of "nothing at stake" because creating blocks or forks is virtually costless compared to PoW's energy expenditure. Without precautions, validators have an incentive to sign multiple competing chains (since there is little immediate cost to do so), which could undermine consensus finality. Modern PoS protocols implement measures to prevent this, such as penalising conflicting votes or using checkpoints and time-based finality to invalidate very old forks (discussed in Chapter 2.4) [29]. In summary, both PoW and PoS aim to secure the blockchain by making it computationally or economically infeasible for a malicious actor to outpace the honest majority. Each comes with distinct mechanics and, as we discuss next, distinct potential attack vectors identified in the literature.

2.2 Sybil Attacks in PoW and PoS

A Sybil attack occurs when an adversary creates many fake identities (nodes) to obtain excessive control over a network [19]. In blockchain consensus mechanisms, this type of attack involves an adversary generating multiple miner or validator identities to seize control of the network's voting power. By design, both PoW and PoS consensus are Sybil-proof in principle because simply multiplying identities does not increase one's effective power unless accompanied by the requisite resources. As John R. Douceur's seminal work on Sybil attacks notes, any decentralised

system must leverage a scarce resource or identity cost to defend against Sybils [19]. Bitcoin's PoW is an elegant solution: it ties identity (the right to influence consensus) to a costly proof-of-work, meaning an attacker with 1% of global hash power, no matter how many separate mining nodes they operate, will still only win $\sim 1\%$ of blocks. Similarly, in PoS, identity is directly linked to stake ownership. Therefore, an attacker with 1% of the total stake cannot gain more than $\sim 1\%$ influence by dividing that stake across multiple validator accounts.

However, the effectiveness of Sybil resistance in PoS ultimately depends on how evenly the resources (stake) are distributed among participants. If an attacker can somehow acquire a large fraction of the total hashing power or total staking tokens, the Sybil resistance is effectively bypassed. This reduces to the majority attack scenario (commonly a 51% attack in PoW or an analogous $> 50\%$ stake attack in chain-based PoS). In PoW, majority attacks can be viewed as an extreme case of Sybil vulnerability: the attacker uses many Sybil identities (or simply controls a large mining pool of honest identities) to collectively hold $> 50\%$ of hash power and consistently override honest miners [27]. In practice, pure identity-based Sybil attacks (where identities alone matter without resources) are not a concern in open blockchains – the main threat is resource accumulation attacks. PoS is sometimes mistakenly said to be "vulnerable to Sybil attacks" because it doesn't burn energy like PoW; however, the requirement to lock up large economic value for influence serves the same role as energy does in PoW [24]. In 2023, Platt and McBurney conducted a comprehensive study of several consensus mechanisms, and they affirmed that PoW and PoS both achieve Sybil resistance by linking identity to a costly resource [28]. They note that PoW's reliance on a physical resource (computational work) limits the rate at which an attacker can expand its influence (due to real-world constraints on hardware and electricity). In contrast, PoS's reliance on an economic resource introduces different risks, such as wealth centralisation or the possibility of exchange breaches yielding large stakes to an attacker [28]. Overall, excluding extraordinary circumstances, neither PoW nor PoS can be subverted by a pure Sybil attack without the attacker also fulfilling the conditions of a majority resource attack.

In summary, Sybil attacks in their pure form are largely mitigated in both systems: PoW makes Sybil identities expensive by requiring proof-of-work [27], and PoS makes them expensive by requiring significant stake deposits [24]. The remaining security questions then focus on scenarios where an attacker commands substantial resources or exploits the protocol in other ways, which leads us to majority attacks, selfish mining, long-range attacks, and social coordination

mechanisms to reject such forks. Because of these safeguards and the difficulty of acquiring a large fraction of stake without detection, major PoS networks (such as Ethereum post-merge) now view long-range attacks as impracticable. That being said, the evolution of PoS consensus protocols has been guided by the warning that naive PoS are susceptible to long-range attacks.

2.3 Selfish Mining and Majority Attacks in PoW

The selfish mining attack represents one of the most well-known vulnerabilities in Proof of Work systems, which Eyal and Sirer introduced in 2014 [20]. A miner or a colluding mining pool performs selfish mining by hiding discovered blocks from broadcast to achieve strategic benefits against honest miners. By keeping a sequence of newly mined blocks private and timing their release strategically, the selfish miner can force honest nodes to waste effort on a stale (doomed) chain, thereby increasing the selfish miner's relative share of rewards beyond what their raw hash power would normally earn [20]. In essence, the selfish miner deliberately creates a secret fork of the blockchain and reveals it opportunistically to outweigh the public chain at key moments, causing honest miners' work on the public chain to be discarded as "orphans." This attack involves complex strategic behaviour, but its profitability can be analysed mathematically. Suppose we denote by α the fraction of total hash power controlled by the attacker and assume the network's propagation dynamics give the attacker some advantage in races (e.g. honest miners sometimes mine on the attacker's chain in a tie). In that case, the attacker's expected fraction of blocks (revenue) $R(\alpha)$ can exceed α for specific values of α. Eyal and Sirer's analysis revealed a critical threshold: when the attacker's hash power exceeds about 25–33% of the total, selfish mining yields higher rewards than honest mining, incentivising the attacker to deviate from the protocol [20]. This result was striking, as it showed that a coalition controlling far less than 51% of the network's power could potentially earn more than its fair share of blocks, undermining the prior assumption that only majority control was dangerous.

Under favourable network conditions for the attacker (for example, if the attacker's blocks propagate significantly faster than others, often modelled by a parameter $\gamma \approx 1$ for tie-breaking), the profitability threshold approaches about 0.25 (25%) [20]. Under more neutral conditions (e.g. $\gamma = 0.5$, meaning honest miners have no bias in how they break ties between competing chains), the threshold is around 33% of hash power [20]. We can illustrate this threshold in quantitative terms. Suppose honest miners always choose the longest chain, and in case of a one-block tie, they randomly choose which fork to build on (a $\gamma = 0.5$ scenario). In this model,

Eyal and Sirer showed that the attacker's relative block reward is approximately:

$$R(\alpha) = \frac{\alpha(1-\alpha)^2 + \alpha^2(1-\alpha)}{1-\alpha(1-\alpha)^2} \tag{2.2}$$

which exceeds the attacker's honest share α once α grows beyond about 0.30 (30%). In the best-case scenario for the attacker (if honest miners always prefer the attacker's fork whenever possible, $\gamma \to 1$), the critical point comes at $\alpha > 0.25$ [20]. Theoretically, a selfish mining pool could capture more than 25% of the block rewards by controlling as little as a quarter of the network's total hash rate. The selfish pool gains economic superiority against honest miners through this advantage. If ignored, the selfish pool would gain more influence through a feedback loop because its higher profitability attracts more miners who join its operation. Ultimately, this could enable the selfish pool to achieve majority control of the network [20]. The figure below showcases the selfish mining process in a very simple manner [26]:

Figure 2.1: Simple Representation of Selfish Mining Process

It is important to note that selfish mining has so far remained mostly a theoretical concern. In practice, no known large-scale selfish mining attacks have been observed on Bitcoin or Ethereum's PoW networks during their many years of operation – real miners prefer the stable, predictable income from honest mining over the risky strategy of selfish mining. Empirical

studies in the 2020s have attempted to detect signs of selfish mining and found, at most, some minor anomalies in smaller cryptocurrencies, but not widespread use of the tactic on major chains [20]. Still, the mere possibility of a sub 51% mining coalition undermining the protocol's fairness was enough to spur research into countermeasures. Bitcoin's protocol was eventually updated to slightly penalise stale blocks (which reduces the gain from selfish mining), and the community closely monitors mining pool sizes to discourage any one pool from approaching these threshold levels.

Beyond selfish mining, majority attacks (attaining $> 50\%$ hash power) in PoW allow an attacker to outright dominate consensus. With majority control, an attacker can create arbitrary forks and extend the blockchain, enabling them to double-spend coins or censor transactions. In this case, the attacker's extended chain cannot be overridden or challenged by honest miners. Fortunately, these kinds of attacks are extremely expensive and are thus uncommon on well-established PoW networks. The more pressing concern has been the sub-majority attacks like selfish mining (and its variants), which exploit incentive issues rather than absolute hash power superiority [20]. The literature has proposed various refinements and monitoring to address these issues. For example, follow-up research refined the selfish mining strategy and its counter-strategies, showing that if the network is made less forgiving to privately mined forks (e.g. through timeliness penalties or improved block dissemination protocols), the effective threshold for profitability can be raised closer to the ideal 50% [20]. In practice, PoW's assumption of an honest majority has been held. However, the existence of these theoretical attacks highlights that incentive compatibility is as important as cryptographic security in consensus protocol design.

2.4 Long-Range Attacks and "Nothing at Stake" in PoS

Proof of Stake introduces different attack considerations since consensus does not depend on expending energy but on owning a stake. One notable vulnerability in PoS (especially early or naive designs) is the long-range attack [29]. In a long-range attack, an adversary tries to rewrite a very old part of the blockchain history – for example, starting from the genesis block or from a point far back in time. This can be feasible in PoS under certain conditions because maintaining a private fork does not require astronomical energy; the attacker only needs the secret keys of stakeholders from that past era. Suppose those stakeholders have since gone offline or sold their coins. In that case, the attacker can use those old keys (perhaps obtained via collusion,

hacking, or purchasing from ex-holders) to create an alternative transaction history from that point onward [29]. The cost to do so is primarily the opportunity cost of having stake locked up on the attack chain, which could be negligible if those coins were no longer economically relevant to the original owners (for instance, coins that were spent long ago or lost). Essentially, PoS by itself lacks an intrinsic, ever-growing cost to prevent a determined attacker from forking the chain's history, especially long after the fact. The "nothing-at-stake" problem exacerbates this situation: validators have little immediate penalty for signing multiple chains, so a malicious validator (or even an idle, indifferent one) could support an attack fork at no direct expense [29].

In contrast, deep history rewrites are practically impossible in PoW. In order to fork the chain from genesis in a PoW system, an attacker would need to re-do all the proof of work that the network accumulated over that entire period, which amounts to an astronomical amount of computation for a mature blockchain. For example, the effort needed to rewrite Bitcoin's historical record from even a few months ago exceeds any current hash power capabilities, which must be maintained consistently across months or years, thus making the task impossible. This is why even PoW attacks like selfish mining can only target the recent tail of the chain – an attacker cannot go back arbitrarily far because the cumulative work on the main chain acts as a nearly insurmountable barrier. PoS lacks this cumulative-work property, so without additional safeguards, an attacker who somehow controls historical stake keys (even if those keys correspond to stake that no longer exists in the current ledger) can generate an alternate chain costlessly and potentially make it longer (in terms of number of blocks or stakeholder signatures) than the main chain.

Long-range attacks were highlighted in the literature as a critical weakness of early PoS proposals [29]. Deirmentzoglou et al. (2019) categorise long-range attacks into several types, such as posterior corruption (where an attacker acquires private keys of old validators after the fact) and stake bleeding (gradually accumulating stake on a forked shadow chain over many epochs) [29]. The common thread is that PoS systems need additional rules or assumptions to prevent an attacker from exploiting the indeterminate security of long-past ledger history. Over the years, several mitigation strategies have been developed and are now standard in modern PoS blockchains:

- **Checkpointing / Weak Subjectivity:** The idea (introduced by Ethereum founder Vitalik Buterin and others) is that nodes should occasionally rely on a checkpoint – a recent block hash or state summary that is widely agreed upon – beyond which the history is considered immutable. In practice, new nodes joining the network are advised to start from a recent checkpoint (e.g. 1000 blocks ago or a certain date) rather than from genesis. This weak subjectivity assumption means that as long as the community agrees on the recent state, an attacker cannot persuade honest nodes to accept a conflicting deep fork that contradicts a known checkpoint [29]. Many PoS networks implement periodic checkpoints (either through the protocol or via social consensus and client software) to limit how far back an attack can go. This does introduce a slight reliance on social coordination (hence "subjectivity"), but it drastically reduces the feasibility of long-range attacks.

- **Slashing Conditions:** As mentioned earlier, PoS protocols like Ethereum's Casper impose penalties if a validator is caught signing two different histories (for example, signing two blocks at the same height on two chains). Suppose an attacker tries a long-range attack, and some honest nodes eventually see that certain validators signed conflicting histories. In that case, those validators' stakes can be slashed (destroyed) in the canonical chain [29]. The threat of losing a large deposit deters rational validators from ever cooperating in such attacks. Of course, slashing is ineffective if the keys used in the attack belonged to validators who no longer have a stake in the current chain (e.g., they sold their coins and left long ago). However, it raises the economic cost for an attacker who still owns some stake since any stake they have on the main chain would be forfeited when exposed as equivocated signers.

- **Finality Gadgets:** Newer PoS implementations use Byzantine Fault Tolerant (BFT) consensus layers to finalise blocks after the fact. For example, Ethereum's recent PoS upgrade (Casper) [15] uses a committee of validators to vote on finalising checkpoints such that once a block is finalised (typically within 10–30 minutes), reverting it would require at least two-thirds of validators to collude (because they would have to undo their finality votes, which honest validators will not do). This means an attacker would need a supermajority of stake at present to revert recently finalised blocks, making long-range attacks impractical unless the attacker already controls the network in real time [29]. Finality creates an ever-growing "anchor" in the chain that an offline attacker cannot

beat without an overwhelming amount of current stake. Many PoS systems (e.g. Cosmos, Polkadot, Ethereum) incorporate such finality gadgets to ensure that after some point, the history is locked in by cryptographic votes.

With these measures, PoS networks have greatly strengthened their security against long-range forks. Nonetheless, it remains a nuanced point that PoS's security, unlike PoW's, is not purely "trustless" across any time span – it may rely on assumptions about an honest majority not just now but also in the recent past, and it often requires some degree of trust in a recent checkpoint or social coordination (hence the term weak subjectivity for PoS security). Due to the possibility of long-range attacks, key management has become important. If old validator keys can be compromised or resurrected, they could be used in an attack. To mitigate the risk, protocols often require that old keys are securely deleted or rotated out over time to limit the attack surface.

To summarise the literature's view: PoS is inherently more vulnerable to history-rewriting attacks than PoW unless protocol safeguards are in place [29]. The low cost of simulating or forking a PoS chain means an attacker's probability of success in a long-range attack is not governed by a steep exponential drop-off as in PoW but rather by whether they can convince nodes to accept their alternate history. Without finality or checkpoints, a determined attacker with enough old keys and capital could create an alternative chain of equal or greater "weight" than the real one (since the weight in a basic PoS might just be the cumulative stake signatures, which the attacker can produce arbitrarily if they control the old signers). The blockchain industry has responded to these challenges by designing PoS protocols with explicit economic finality and social coordination mechanisms to reject such forks. In contemporary Proof of Stake networks, such as Ethereum, long-range attacks are widely regarded as impractical. This is primarily because of sophisticated protective mechanisms and the significant difficulty attackers face when attempting to acquire substantial amounts of stake covertly. However, the fact that early, basic PoS designs were vulnerable to long-range attacks has been a useful warning. This has significantly influenced the ongoing refinement and development of contemporary PoS consensus protocols.

2.5 Comparative Insights from the Literature

Having examined each attack type in isolation for PoW and PoS, we can now compare how these consensus mechanisms fare relative to each other.

- **Sybil/Majority Attack Resistance:** PoW and PoS require an attacker to command majority resources (hash power or stake) to control the chain outright. PoS achieves security through economic mechanisms, requiring participants to acquire and lock up stake. In contrast, PoW relies on the expenditure of physical resources, such as investing in specialised hardware and consuming significant amounts of energy. Literature suggests that for established, widely decentralised networks, both scenarios are extremely costly and thus unlikely [28]. However, PoW attacks might be more transient in nature: hash power can sometimes be rented or redirected for short periods, as seen in attacks on smaller PoW coins, whereas PoS attacks require a long-term investment in the asset (which would likely crash in value if the attack undermines the system). Platt and McBurney (2023) concluded that no consensus mechanism is completely immune to Sybil or majority attacks. However, both PoS and PoW significantly raise the threshold of resources required for a successful attack [28]. Historically, PoW has demonstrated greater resilience against such threats, especially within major networks like Bitcoin. On the other hand, PoS is newer but is designed with additional safeguards to make sustained majority attacks detectable or economically self-defeating (e.g., PoS attackers can be slashed or socially ousted since their stake exists within the system's economy and can be tracked).

- **Selfish Mining and Incentive Compatibility:** This category of attack is specific to PoW's longest-chain rule and the block propagation dynamics. PoS longest-chain protocols could, in theory, suffer analogous attacks (sometimes termed selfish validating or stake grinding if a staker tries to privately build a fork or manipulate randomness to their advantage). However, the dynamics differ because in PoS, the probability of extending your own fork does not increase straightforwardly with stake in the way it does with hash power (especially if finality is involved or if leader selection is truly random and unpredictable) [22]. Research such as Guru et al. (2023) has surveyed attacks on various consensus protocols and notes that PoW's selfish mining remains the classic example of a sub-majority attack exploiting the incentive structure [20] [22]. In PoS, the "nothing-at-stake" issue was the bigger concern, which has been largely mitigated by protocol design (e.g. through slashing and finality as discussed). Thus, one might say PoW has a

known weakness in incentive compatibility (selfish miners can gain more than their fair share under certain conditions). In contrast, properly designed PoS aims to be incentive-compatible by construction (deviating is made unprofitable or penalised). Importantly, selfish mining in PoW can potentially be detected by observing unusual rates of orphaned blocks or fork patterns, and it is self-limiting unless the attacker is near the threshold. Consensus adjustments have addressed PoS's equivalent issues (like creating multiple forks), so modern PoS chains do not reward equivocation in the way early designs might have.

- **Long-Range Attack Exposure:** Here, PoW clearly has the upper hand – a long-range (deep history) attack on PoW is considered "computationally prohibitive" and not a realistic concern [29], whereas PoS must actively guard against such attacks. The survey by Deirmentzoglou et al. (2019) emphasises that long-range attacks are a unique challenge for PoS protocols and must be addressed for PoS to be secure in the long term [29]. As a result, modern PoS systems incorporate checkpointing and finality as described above. This points to a philosophical difference: PoW's security is memoryless and relies only on current hash power. In contrast, PoS's security has a temporal dimension (recent stake distribution and recent honest behaviour matter more than ancient history). New PoS nodes need some trusted recent snapshot to know the correct chain, whereas PoW nodes can, in theory, sync from genesis by always trusting the heaviest (work-wise) chain.

In conclusion, the literature provides a balanced view because neither PoW nor PoS is inherently better in all security aspects. Although PoW has a well-established security model based on physical laws and energy costs, which makes some attacks like long-range history rewrites nearly impossible without an absurd amount of processing power, it also has inefficiencies and a potential incentive flaw (selfish mining) that could threaten its decentralisation if miners behave greedily [20]. PoS improves efficiency and can leverage economic penalties to discourage misbehaviour. However, it requires more complex protocol rules to defend against attacks that PoW naturally shrugs off (like costless forking in the absence of work) [29]. These trade-offs highlight the fact that there is no one-size-fits-all solution, and every strategy requires careful security engineering to address its flaws and weaknesses. Current research and practical experimentation are actively investigating hybrid and innovative approaches that leverage the advantages of established consensus mechanisms. By synthesising my findings, this report aims to contribute insights intended to support future efforts to "secure the chain" against both

established and emerging threats.

Chapter 3

Requirements and Specification

3.1 Brief

This chapter defines the objectives and criteria the simulation framework must meet through clear and detailed specifications. The system needs to fulfil essential features and capabilities, which the chapter first defines as key functional requirements. Alongside these, a set of non-functional requirements ensures the quality, usability, and robustness of the solution. The requirements define exactly what the project intends to accomplish through their combined specifications. Finally, the chapter clarifies the scope and key assumptions underlying the simulation, ensuring that the design and implementation in later chapters can be properly focused.

3.2 Functional Requirements

ID	Requirement	Description
FR1	PoW and PoS models	Model PoW (hash-based block production, longest chain rule) and PoS (stake-based validator selection per time slots) consensus mechanisms.
FR2	Sybil Attack Scenario	Simulate Sybil attacks for PoW and PoS, comparing identity-based vs. resource-weighted voting
FR3	Long-Range Attack Scenario	Simulate PoS long-range attacks from historical forks, determining attacker success in rewriting blockchain history within a specified round limit.
FR4	Selfish Mining Attack Scenario	Simulate PoW Selfish Mining attack per Eyal and Sirer's strategy, tracking strategic block withholding and attacker block-reward advantage [20].
FR5	Parameterisation	Enable adjusting of key simulation parameters (e.g., attacker resource fraction, Sybil identities, chain gaps, number of honest nodes).
FR6	Utilise Multi-run Monte Carlo Simulations	Use multi-run Monte Carlo simulations to gather statistically robust results by averaging multiple stochastic executions [5].
FR7	Results Logging	Automatically log critical metrics from simulations (e.g., attacker success rate, chain lengths, block share) in a structured format (e.g., CSV).
FR8	Results Visualisation	Generate basic visual plots for interpreting simulation outcomes, validating trends, and clearly demonstrating results.

3.3 Non-Functional Requirements

ID	Requirement	Description
NFR1	Accuracy and Validity	Simulation outcomes must align with theoretical expectations. Under honest conditions, outcomes should favour the honest majority; deviations should be explainable by randomness or known limitations [20][18].
NFR2	Efficiency	Simulator should efficiently handle extensive simulations (e.g., 1000+ blocks/slots and 20–100 Monte Carlo repetitions).
NFR3	Modularity and Clarity	Codebase must be modular, clearly separating consensus logic, attack strategies, and results handling, facilitating easy maintenance, testing, and future extensions.
NFR4	Reproducibility	Support pseudo-random but reproducible runs via configurable random seeds to ensure results can be replicated for verification and debugging.
NFR5	Usability for Experimentation	Utilise an intuitive configuration mechanism (e.g., CLI) enabling users to conveniently execute various attack scenarios with flexible parameters, supporting extensive experimentation.
NFR6	Data Integrity	Logged simulation results must accurately and reliably reflect events.

3.4 Scope and Assumptions

The project scope is carefully limited to consensus-level simulation and does not attempt to reproduce a full blockchain with all real-world factors. The key assumptions and constraints are:

- We ignore transaction-level details and network communication complexities. The simulation operates in simplified rounds of block creation. This is justified since the attacks in question (Sybil, selfish mining, long-range) primarily threaten the consensus mechanism rather than transaction validity or networking. For example, network latency and propagation are assumed to be ideal (instantaneous) except where an attack explicitly withholds blocks (selfish mining).

- We assume a fixed set of honest participants and (at most) one attacker. Sybil attacks are modelled by one attacker controlling many identities rather than multiple independent attackers. We also assume the total resource in the system (hash power or stake) remains constant during the simulation, divided between honest and adversary.

- In the PoW model, mining difficulty is assumed constant, and time is abstracted in terms of block counts rather than real-time. This allows for a focus on the probabilistic race aspect of mining. Similarly, the PoS model uses a fixed number of slots with equal weight per slot; we do not model dynamic stake changes or coin transactions over time.

- We assume that under normal operation, honest nodes control the majority of resources ($> 50\%$ of hash power or stake) in order to maintain security [17]. The attack scenarios then examine edge cases as the attacker's share grows. We do not address outright 51% attacks in PoW or $> 50\%$ stake attacks in PoS, as those trivially break the system by assumption; instead, the focus is on sub-majority attacks (selfish mining can succeed with $< 50\%$ power [20], and long-range attacks exploit PoS-specific weaknesses even with $< 50\%$ current stake [18]).

- Each attack strategy is implemented according to a well-known method from research. For example, the selfish miner follows the state-machine strategy from Eyal & Sirer [20] (including when to publish withheld blocks). We assume the attacker commits to the attack strategy and does not deviate or mix strategies. This allows us to evaluate the best-case effectiveness of each attack in isolation.

- Social or economic defences (e.g. community detection of attacks, dynamic adjustments like kicking out Sybil identities, etc.) are outside the scope of the simulation. The study is focused on technical protocol-level behaviour. Mitigations like checkpointing in PoS or difficulty readjustments are noted in the literature [18] but are not implemented; instead, we discuss their impact in analysis.

Chapter 4

Design

This chapter describes the high-level design of the simulation framework developed to meet the specified requirements.

4.1 Simulation Architecture

4.1.1 Core Classes and Components

The design contains an abstract `AttackScenario` base class that serves as a general simulation scenario, together with two simple data classes for `Node` and `Block`. The three attack types (Sybil, Long-Range, and Selfish Mining) are implemented as subclasses of `AttackScenario`, each overriding a `run()` method that encapsulates the scenario's logic. The object-oriented design structure provides modularity while enabling the inheritance of common functionality, such as running multiple trials and logging results, thereby reducing code duplication.

The `Node` class defines network participants with attributes including hash power, stake values, and attacker status. The `Block` class defines mined and forged blocks through its basic properties, including height and miner identity because transaction details remain unmodelled. The established abstractions provide sufficient capability to model block distribution across miners during chain expansion.

4.1.2 Simulation Execution Engine

The framework orchestrates simulations via the `AttackScenario` interface. A scenario instance is initialised with the needed parameters (per FR5, e.g., attacker power, number of rounds, etc.). Then, the `run()` method can be called to simulate one realisation of that scenario. To fulfil FR6 (Utilise Multi-run Monte Carlo Simulations), the simulation includes a method that repeatedly executes `run()` and aggregates the outcomes. Each run yields a dictionary of result metrics (fulfilling FR7), which are stored internally and later used for analysis or visualisation purposes. The framework includes built-in functionality for result logging to CSV files and automatic result plotting using matplotlib for quick feedback [7]. These features enable simple experiment execution and observation, thus satisfying NFR5 usability requirements.

4.2 Attack Scenarios

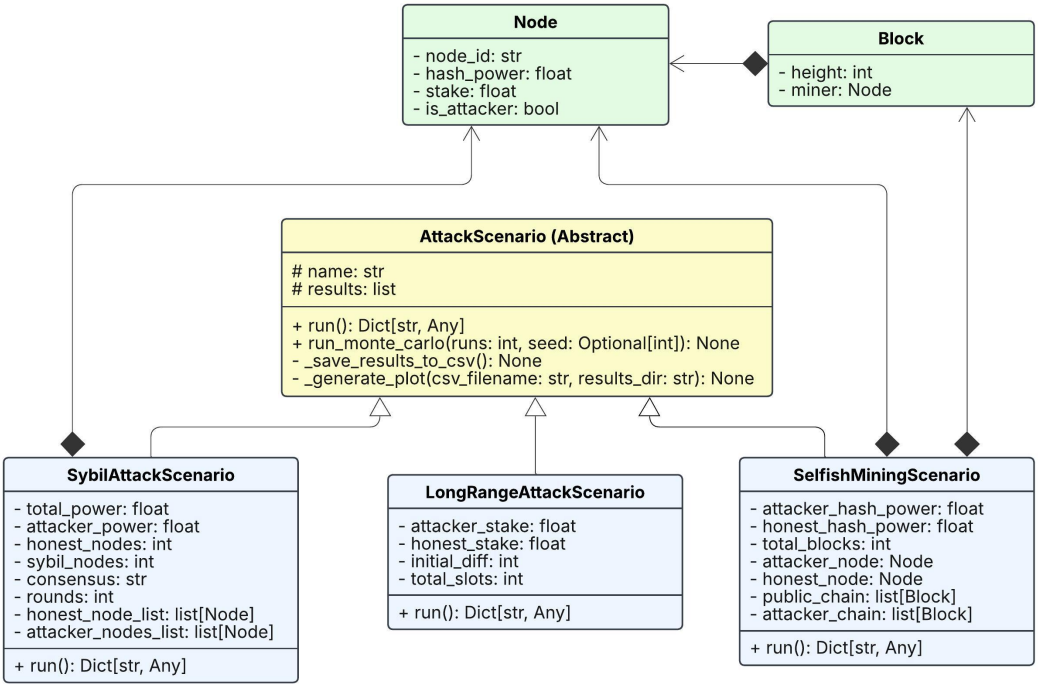

Figure 4.1: UML Class Diagram for Attack Scenarios

4.2.1 Sybil Attack Scenario (PoW and PoS)

Design rationale: A pure Sybil attack targets a system where influence is based on identity count rather than resources [19, 27]. Real PoW/PoS blockchains are, by design, Sybil-resistant (mining power or stake is required to influence blocks, not just identities) [27]. However, this

scenario is included to quantitatively illustrate why PoW/PoS use weighted voting [25]. The `SybilAttackScenario` class is designed to simulate two parallel consensus selection methods:

1. **Identity-Based Selection:** Each node (honest or Sybil) gets an equal chance to be selected to produce the next block (or win a round), regardless of resources. This models a naive protocol with no Sybil protection.

2. **Resource-Weighted Selection:** Nodes are chosen with probability proportional to their resource (hash power or stake), reflecting a proper PoW or PoS mechanism.

To set this up, the scenario initialisation takes total resource (normalised to 1.0), attacker's resource fraction, number of honest nodes, and number of Sybil identities (FR2). It then creates a list of honest `Node` objects, dividing the honest resource evenly among them, and a list of attacker `Node` objects (Sybil identities), dividing the attacker's resource equally. In the `run()` loop for a fixed number of rounds (each round analogous to a block or leadership slot):

- **Unweighted mode:** The winner is chosen uniformly at random from all nodes (using `random.choice` over the list of nodes). We increment `attacker_wins_identity` if the chosen node is one of the attacker's Sybil identities.

- **Weighted mode:** We simulate a fair PoW/PoS selection by picking a random number in (0, `total_power`) and selecting the node whose cumulative weight exceeds that threshold. This effectively selects a node with probability equal to that node's fraction of total power/stake. We increment `attacker_wins_weighted` if that selected node is controlled by the attacker.

After all rounds, we compute the fraction of rounds the attacker won in each mode (identity-based vs. weighted). The expected result is that in identity-based mode, an attacker controlling, say, 30% of the resource but splitting into many identities can win much more than 30% of the rounds (potentially exceeding 50% if they create an enormous number of identities, overwhelming the few honest identities). In contrast, in weighted mode, the attacker's win rate should approximately equal their 30% resource share (aside from statistical variance). This scenario effectively shows the Sybil vulnerability: without resource weighting, consensus can be subverted cheaply by creating fake identities [19, 27]. By designing the simulation to output both *"identity win rate"* and *"weighted win rate"* for various attacker fractions, we directly satisfy FR2 and provide quantitative evidence for the necessity of Sybil resistance in blockchain

protocols.

4.2.2 Long-Range Attack Scenario (PoS)

Design rationale: Long-range attacks exploit PoS's weakness, which is that an attacker holding old private keys from a past state can attempt to fork the chain without current resource cost [18]. The `LongRangeAttackScenario` class models a simple form of this: We assume at some past checkpoint (fork point), the attacker had a certain stake fraction. The honest chain has continued and is ahead by d blocks (the `initial_diff` parameter). Now, the attacker tries to privately generate an alternative chain from that fork, hoping to catch up. Our design simulates slot-by-slot block production for a given number of slots (time steps) after the fork:

The following pseudocode (Algorithm 1) succinctly summarises the logic of this probabilistic race scenario:

Algorithm 1 Long-Range Attack Probabilistic Race

attacker_chain_length $\leftarrow 0$
honest_chain_length $\leftarrow d$ ▷ Honest chain starts d blocks ahead
success $\leftarrow$ False
for $t \leftarrow 1,$ total_slots **do**
  if random() $< \frac{\text{attacker\_stake}}{\text{attacker\_stake+honest\_stake}}$ **then**
    attacker_chain_length $\leftarrow$ attacker_chain_length $+ 1$ ▷ Attacker wins slot
  else
    honest_chain_length $\leftarrow$ honest_chain_length $+ 1$ ▷ Honest validators win slot
  end if
  if attacker_chain_length $\geq$ honest_chain_length **and** success = False **then**
    success $\leftarrow$ True ▷ Attacker catches up
  end if
end for
return success

- Each slot, either the honest chain or the attacker's fork produces a new block. We assume the attacker can always attempt a block each slot (using their old stake), and honest validators produce a block each slot as well (the model is effectively one block per slot, whoever wins that slot).

- The probability that the attacker wins a slot (i.e., their fork extends) is set to their stake fraction at the fork (e.g., if the attacker had a 30% stake at that point, in each slot, they have a 30% chance to create the block, versus a 70% chance an honest node creates it on the main chain). This is a simplifying assumption akin to Ouroboros-style random selection proportional to stake [23].

- We iterate this for the specified number of slots (e.g., 100 or 1000). The attacker's chain length starts at 0 (just the fork point), and the honest chain starts at length d (the head-start). We then simulate slot outcomes. If attacker wins a slot, `attacker_chain_length`++; if honest wins, `honest_chain_length`++.

- At the end, we check if `attacker_chain_length` $\geq$ `honest_chain_length`. If so, the attacker's fork has caught up or surpassed the honest chain, meaning a successful long-range attack (the network could be convinced to adopt the attacker's history). If not, the attack failed in this trial.

This design captures the essence of a long-range attack in a probabilistic race model. It abstracts away details like validator rotation or slashing, focusing purely on the probability of eventually overtaking given an initial deficit. We run many trials (Monte Carlo) for each set of parameters, and then we estimate the probability of success of the attack. For example, we might vary the attacker's stake fraction from 10% to 90% (FR5) and observe how success probability increases from near 0 to near 100%. We expect that if the attacker's stake is small and the honest chain has even a modest lead, the chance of catching up is very low (akin to a random walk with negative drift). But as the attacker's stake grows or the honest lead (d) is small, the attack becomes more feasible. FR3 is satisfied by this scenario implementation, and it aligns with literature insights: PoW is practically immune to such far-past fork takeovers (an attacker would need to redo enormous work [18]), whereas PoS is exposed unless protocols add defences like checkpoints [18]. Our simulation provides concrete numbers for this risk in a simplified setting.

4.2.3 Selfish Mining Scenario (PoW)

Design rationale: We base this on the well-known selfish mining strategy [20], where an attacker withholds blocks to gain an advantage. The `SelfishMiningScenario` class encapsulates this logic by setting up two parallel blockchain states: the public chain (followed by honest miners) and the attacker's private chain. The simulation iteratively generates the next block by randomly deciding whether the honest miners or the attacker finds it, proportional to their hash power split (this uses a random draw, modelling the race with probability for the attacker finding the next block). The core algorithm follows the states described in the literature [20]:

The attacker's decision-making process is succinctly summarised by the following pseudocode in Algorithm 2:

Algorithm 2 Selfish Mining Attack Strategy (PoW)

```
lead ← 0                                              ▷ Attacker's private lead
while not end_of_simulation do
    if random() < α then                              ▷ Attacker finds next block
        Add block to attacker's private chain
        lead ← lead + 1
    else                                              ▷ Honest network finds next block
        if lead = 0 then
            Add honest block to public chain
            lead ← 0
        else if lead = 1 then
            Publish one private block to public chain  ▷ Chains tied
            lead ← 0
        else                                          ▷ lead ≥ 2
            Publish all but one private blocks to public chain
            lead ← 1
        end if
    end if
end while
Publish any remaining private blocks to public chain
```

- If the attacker finds a block, it is added only to their private chain (increasing their lead).
- If an honest miner finds a block, the attacker reacts based on their current lead:
  - If the attacker was even, the honest block is adopted, and the attacker's private chain (if any) is discarded.
  - If the attacker immediately publishes their withheld block to tie with the honest block, creating two competing blocks of equal height (one from the attacker, one honest).
  - If the attacker publishes just enough blocks to remain one block ahead of the public chain (keeping one block still secret).
- These rules ensure the attacker maximises the chance their chain becomes longer and accepted without giving honest miners sufficient time to surpass them.
- The simulation continues until a fixed number of blocks have been added to the public chain (e.g., 100 or 1000 blocks as required, excluding the genesis block). At that point, any remaining unpublished attacker blocks are released to finalise the comparison.
- The design tracks metrics such as (the number of blocks in the public chain mined by the attacker) and computes the attacker's fraction of the main chain. This directly addresses the goal of measuring the impact of selfish mining (FR4). By running this scenario for various attacker hash powers, we collect data on how the attacker's block share compares to their hash power, confirming a known theoretical curve (where, above a certain threshold, the attacker's share exceeds).

The design thus captures the subtle incentive advantage a selfish miner can gain, consistent with the original attack description.

4.3 Data Management and Analysis

4.3.1 Data Collection, Logging, and Visualisation

For each attack scenario, the design produces key metrics in a structured format, addressing FR7. The `AttackScenario` base class handles writing results from all simulation runs into CSV files and generates relevant plots automatically. The results of each scenario present parameters

together with key outcomes of interest. For example:

- **Selfish mining scenario** outputs the attacker's hash power fraction and the fraction of blocks they mined. Aggregating these data over multiple runs or varying α values allows plotting comparisons against the fair share line ($y = x$), as illustrated in Figure 6.4 (Results and Analysis chapter).

- **Sybil attack scenario** outputs the attacker's resource fraction and two metrics: identity-based and weighted win rates. These metrics can be plotted to highlight the divergence caused by Sybil identities.

- **Long-range attack scenario** outputs include the attacker's stake fraction, initial difference d, and a success flag for each run. By averaging the multiple runs of success flags, we produce an empirical success probability based on specific conditions.

Automating logging and plotting promotes clarity and efficiency (NFR5), reducing manual processing effort and enables fast verification of simulation behaviours. It also enhances traceability (NFR6: data integrity), as the raw CSV files provide a transparent way to validate calculations and ensure consistency in plotting, minimising human errors in analysis.

4.4 Design Evaluation

4.4.1 Clarity and Extensibility

The class structure's simplicity, which defines clear purposes for each class and separates attack logic, ensures the design remains straightforward to comprehend and expand. Adding new attack scenarios (e.g., a 51% attack or network partition scenario) involves only creating a new subclass of `AttackScenario` with the required `run()` method, without altering existing classes. This design approach fulfils the requirement of NFR3 (modularity).

The parameters for each scenario are isolated as attributes within their respective scenario classes, providing a clear understanding of variable effects on experiments. The CLI or scripting configuration creates attack scenarios with specific parameters, directly executing them through a clean interface that minimises misconfiguration risks and ensures reproducibility (NFR4), as results depend solely on parameter sets.

4.4.2 Design Justification and Trade-offs

The chosen design balances realism and simplicity. We adhere to known algorithms from research for the attack implementations while stripping away extraneous complexities (e.g., we don't simulate transaction propagation or dynamic validator sets) to focus on the research question [20][18]. The data structures and flow are kept minimalistic (e.g., using simple lists to track chains or nodes) but this is intentional to allow running large numbers of simulations quickly (NFR2). Any more complex design (such as a full blockchain node software) would be unnecessary for the purpose and hinder meeting the time/efficiency requirements.

Chapter 5

Implementation

5.1 Brief

This chapter details the implementation of a Python-based simulation framework used to model PoW and PoS blockchains and to evaluate their resilience against Sybil attacks, long-range attacks, and selfish mining attacks. In this chapter, we focus on the developed core algorithms and logic structures rather than high-level design decisions (discussed in the Design chapter). The code is written in Python 3, with simple object models for blockchain entities and randomised procedures to simulate consensus processes. Each attack scenario was implemented as a self-contained module with a common interface to enable repeated simulations and data collection for analysis. In the following sections, we describe the implementation of the simulation framework and each attack scenario and illustrate critical parts of the code to clarify how the algorithms work.

5.2 Simulation Framework Overview

A basic blockchain simulation framework was developed to enable different attack simulations. The framework introduces foundational classes for nodes and blocks and a generic `AttackScenario` interface, which each specific attack scenario extends. These components establish the basis for running simulations under controlled and repeatable conditions.

5.2.1 Node and Block Representation

The system uses a `Node` class to model participants in the blockchain network. Each `Node` object contains a unique identifier together with resource parameters representing hash power for PoW systems or stake for PoS systems. A boolean flag indicates whether the node is a malicious attacker or honest.

The `Block` class represents mined or forged blocks within the blockchain. The `Block` class contains two attributes: the block height and the `Node` reference that produced it. The model does not include block contents such as transactions. Instead, it focuses exclusively on consensus mechanisms and chain growth. The simplified class structure enables us to track block producers and monitor chain lengths. The code snippet below demonstrates these class definitions:

```python
class Node:
    def __init__(
        self,
        node_id: str,
        hash_power: float = 0.0,
        stake: float = 0.0,
        is_attacker: bool = False,
    ):
        self.node_id = node_id
        self.hash_power = hash_power
        self.stake = stake
        self.is_attacker = is_attacker

class Block:
    def __init__(self, height: int, miner: Node):
        self.height = height
        self.miner = miner
```

Code Snippet 5.1: Python class definitions for Node and Block

In this implementation, a node's influence in PoW or PoS is determined by its `hash_power` or `stake` attribute, respectively, and not by the number of identities it controls. Blocks carry only the information needed to attribute them to a miner. This design choice aligns with the goal of focusing on consensus behaviour and attack impact without extraneous blockchain details.

5.2.2 Attack Scenario Base Class

Each attack (Sybil, long-range, and selfish mining) is implemented as a subclass of a common `AttackScenario` base class. This base class establishes a common interface and utilities for running simulations and collecting results. The base class defines an abstract `run()`

method which needs to be implemented by each subclass with scenario-specific logic, and a `run_monte_carlo()` method to execute multiple runs and aggregate results. By centralising repeated-run logic, the implementation ensures consistency across experiments and satisfies the requirement for reproducible Monte Carlo simulations. For example, the `run_monte_carlo()` method calls the scenario's `run()` method repeatedly (with fresh random seeds for each run) and stores the outcome of each run in a results list. After running the specified number of trials, it writes the aggregated results to a CSV file and can trigger simple automated plotting of results. The automation makes the analysis easier and fulfils the project's non-functional requirements for result collection and visualisation (e.g., logging outcomes to CSV and generating charts for each scenario). The `AttackScenario` base thus encapsulates common functionality, allowing each concrete scenario class to focus on implementing the attack-specific logic within its `run()` function.

5.2.3 Sybil Attack Simulation

The first scenario to be implemented was the Sybil attack simulation. It shows that the mere existence of identities (Sybil nodes) does not give additional influence in properly designed PoW/PoS systems. This scenario compares two modes of leader selection in a simulated consensus round: one unweighted (identity-based) and one resource-weighted (based on hash power or stake). The goal is to demonstrate that in a naive identity-based protocol, an attacker who creates many identities can increase their chance to control the consensus. In contrast, in a resource-weighted protocol (as in PoW or PoS), the attacker's influence remains proportional to their resource fraction, not the number of identities.

5.2.4 Scenario Setup

The Sybil attack simulation is configured with the following parameters: the total resource in the network (normalise to 1.0 for convenience), the attacker's share of this resource, the number of honest nodes, and the number of Sybil nodes (identities) controlled by the attacker.

Using these inputs, the simulation initialises a list of `Node` objects representing the honest participants and the attacker's Sybil identities. Honest nodes collectively possess `total_resource - attacker_resource`, evenly distributed among the honest nodes (each honest node thus receives an equal share of hash power or stake).

The attacker's total resource is equally divided among the specified Sybil nodes. For example,

if the attacker controls 30% of the total mining power but splits it into 10 Sybil identities, each identity receives 3% of the hash power. This initialisation reflects the scenario of an attacker appearing as multiple distinct miners or validators without increasing total resources.

The code snippet below illustrates how the simulation allocates resources to create the node list for the scenario:

```
# Initialise honest nodes
honest_power_total = total_power - attacker_power
honest_power_per_node = honest_power_total / honest_nodes if honest_nodes > 0 else 0.0

nodes = []
for i in range(honest_nodes):
    nodes.append(Node(f"Honest{i+1}", hash_power=honest_power_per_node, is_attacker=False))

# Initialise attacker (Sybil) nodes
attacker_power_per_id = attacker_power / sybil_nodes if sybil_nodes > 0 else 0.0

for j in range(sybil_nodes):
    nodes.append(Node(f"Attacker{j+1}", hash_power=attacker_power_per_id, is_attacker=True))
```

Code Snippet 5.2: Sybil Attack Simulation Node Initialisation

The `total_power` variable shows the complete computational power (or total stake) in the network, while `attacker_power` is the portion controlled by the attacker. We create a list of `Node` objects, where some nodes are labelled `"Honest1"`, `"Honest2"`, etc., each with an equal `hash_power` share, and other nodes are labelled `"Attacker1"`, `"Attacker2"`, etc., each marked with `is_attacker=True` and having a fraction of the attacker's power. For a PoS Sybil scenario, the logic is identical except for using the `stake` attribute instead of `hash_power`—the implementation supports both via the same structure by simply interpreting the resource as hash power or stake based on a parameter. At this stage, the Sybil attacker has multiplied their identities, but each identity individually has only a small fraction of the total power.

5.2.5 Consensus Round Simulation

The `SybilAttackScenario.run()` method then simulates a series of consensus rounds (analogous to successive block proposals or leader elections) and records how often the attacker wins in two situations: (1) if the protocol were identity-based (not resistant to Sybil attacks), and

(2) if the protocol is resource-based (the actual PoW/PoS case). Each round represents a single opportunity for a node to create the next block. The algorithm for each round is as follows:

Unweighted selection: One node is chosen uniformly at random from the list of all nodes, giving every individual identity an equal chance. If the chosen node is one of the attacker's Sybil identities, we count that as an attacker *win* under the identity-based scheme.

Weighted selection: One node is chosen at random with probability proportional to its resource. This is implemented by picking a random number in the range $[0, \text{total\_power})$ and iterating through the nodes, accumulating their hash power (or stake) until the threshold is reached; the node that crosses the threshold is selected. This method ensures that the probability of picking a particular node is equal to that node's fraction of the total power. If the selected node is controlled by the attacker, we count it as an attacker *win* under the weighted scheme.

The code snippet below illustrates the core logic of a single round in the simulation, showing both the unweighted and weighted selection processes:

```
attacker_wins_identity = 0
attacker_wins_weighted = 0
total_power = self.total_power

for _ in range(self.rounds):
    # Identity-based (unweighted) selection
    winner_identity = random.choice(self.all_nodes)
    if winner_identity.is_attacker:
        attacker_wins_identity += 1

    # Resource-weighted selection (PoW or PoS)
    pick = random.random() * total_power
    cumulative = 0.0
    winner_weighted = None
    for node in self.all_nodes:
        weight = node.hash_power if self.consensus == "pow" else node.stake
        cumulative += weight
```

```
        if pick <= cumulative:
            winner_weighted = node
            break
    if winner_weighted and winner_weighted.is_attacker:
        attacker_wins_weighted += 1
```

Code Snippet 5.3: Sybil Attack Round Simulation

In the unweighted case, we use Python's `random.choice` to pick uniformly from the list of nodes. In the weighted case, we generate a uniform random number and iterate through the nodes. Then we sum their weights until the random threshold is reached. This technique is equivalent to drawing a node according to a probability distribution defined by the nodes' resource fractions.

After running the desired number of rounds, the simulation computes the fraction of rounds won by the attacker in each mode (simply `attacker_wins_identity / rounds` and `attacker_wins_weighted / rounds`). These outcomes illustrate the effect of the Sybil attack: in identity-based mode, an attacker controlling many identities can win disproportionately often, whereas, in the weighted mode, their win rate should approximately equal their resource fraction [11].

By comparing these two metrics, the implementation directly measures the degree to which resource weighting (as in PoW/PoS) provides Sybil resistance. The results from this scenario (discussed in the Results and Analysis chapter) confirm that under proper PoW/PoS rules, the attacker's success rate remains near their resource percentage. In contrast, a purely identity-based system would be highly vulnerable (the attacker's success climbs with each additional Sybil identity, approaching the fraction of identities they control).

5.3 Long-Range Attack Simulation

The long-range attack scenario targets Proof-of-Stake systems and models an attacker who attempts to rewrite the blockchain history from a far-past point (e.g., a point at which they held a large stake, which they might have since sold). In a long-range attack, the attacker uses historical stake (which might no longer be owned by them in the current chain) to secretly forge an alternative chain starting from some block in the past. The attacker aims to generate a fork that matches or exceeds the length of the genuine honest chain. At this point, they could

present this fork as the "true" chain while attempting to reverse transactions and compromise the ledger's security. This scenario is specific to PoS because PoW attacks cannot effectively reuse past hashing power without continuously expending energy, whereas in PoS, an old private key with a formerly large stake could sign a new chain at virtually no cost if not properly mitigated.

5.3.1 Scenario Setup

In the simulation, we assume a fork point in the past where the attacker begins their alternative chain. At the moment of the fork, the honest chain is ahead by d blocks (this d is the initial difference representing how far the honest chain has progressed beyond the fork point when the attacker starts the attack). The attacker is assumed to have had a certain fraction of the total stake at that fork point (for example, 30% of all stake), which they now use to try to create new blocks on their private fork. The honest network collectively has the remaining stake (e.g., 70% in this example).

For simplicity, the simulation aggregates the honest stakeholders into a single representative entity and the attacker's stake into another, effectively modelling the competition as a race between two participants: an honest super-node with stake share $1 - \text{attacker\_fraction}$ and an attacker node with stake share equal to the attacker's fraction. This abstraction is reasonable because, in a PoS chain, the probability of any given stakeholder producing a block in a slot is proportional to their stake; many honest nodes, each with a small stake, can be lumped together for probability calculations versus one attacker with an equivalent total stake—the outcome distribution remains the same.

5.3.2 Fork Race Simulation

The attack is simulated over a fixed number of time slots (rounds) after the fork point. Each slot represents a round of block creation opportunity (for example, a slot could correspond to a single PoS block interval). In each slot, either the honest chain or the attacker's chain will produce a block, but not both. We assume the attacker dedicates each slot to extending their private fork (using their historical stake to try to forge a block), and the honest network also tries to extend the main chain in that slot.

Effectively, there is a competition every round to determine which side wins the block for that time slot. The probability that the attacker wins a given slot is set equal to the attacker's

stake fraction at the fork. This models a random leader election or block proposal mechanism proportional to stake (similar to Ouroboros' lottery approach in Cardano, where each slot has a chance for a stakeholder to be the leader, proportional to their stake share) [12]. For example, if the attacker had 0.3 (30%) of the stake at the fork, in each slot, there is a 30% chance that the attacker's chain gains a block (attacker wins the slot) and a 70% chance that the honest chain gains a block.

The simulation iterates slot by slot whilst updating the lengths of the attacker's fork and the honest chain. The simulation begins with the attacker's fork length at 0 because the fork starts at the point of the attack with no new blocks, and the honest chain's length is d (the head start of the honest chain). Then, for each slot:

- A random draw decides the winner. If the attacker is chosen (with probability equal to the attacker stake fraction), the attacker's chain length increases by one (the attacker successfully mines a block on the private fork), while the honest chain does not grow in that slot. If the honest side is chosen, then the honest chain length increases by one (an honest block is added to the main chain), and the attacker's chain remains the same length in that round.

- This process repeats for the predetermined number of slots. The difference in chain lengths (attacker length minus honest length) evolves as a random walk: it starts at $-d$ (since the attacker is d blocks behind at the start) and changes by +1 or –1 each round, depending on who wins each slot.

The code below shows the core loop of this long-range attack simulation. It uses a probability p equal to the attacker's stake fraction to decide which chain to extend each round, and it tracks the chain lengths as described:

```
# Initialise chain lengths (honest chain starts d blocks ahead of attacker)
attacker_chain_length = 0
honest_chain_length = self.initial_diff  # start honest chain with an initial lead

# Simulate each time slot after the fork
for _ in range(self.total_slots):
    # Each slot, either attacker or honest adds a block
    if random.random() < (self.attacker_stake / self.total_stake):
        attacker_chain_length += 1
    else:
        honest_chain_length += 1

# The attack succeeds if attacker's chain length catches up to honest chain
success = attacker_chain_length >= honest_chain_length
```

Code Snippet 5.4: Long-Range Attack Round Simulation

In this snippet, `initial_diff` (d) is the number of blocks the honest chain was ahead of at the time of the fork. The probability `p_attacker` is calculated as the attacker's stake fraction (the code uses `attacker_stake/(attacker_stake + honest_stake)`, which is equivalent to the attacker's percentage of total stake). A uniform random number determines whether the attacker wins the slot. If the attacker wins, we increment `attacker_chain_length`; if not, we increment `honest_chain_length`. The simulation uses a loop for a fixed `total_slots` iterations to model the passage of time after the fork. During each iteration, we also check if the attacker's chain has caught up with or surpassed the honest chain (i.e., `attacker_chain_length` $\geq$ `honest_chain_length`). If so, we mark the run as a success for the attacker. This condition being true means the attacker's alternate history is at least as long as the honest history, implying it could potentially be accepted as the legitimate chain by the network.

After all slots have been simulated, we determine the final outcome of the attack. In practice, if the attacker manages to catch up at any point, that would be a successful long-range attack (since the attacker could then reveal the alternate chain and challenge the main chain's validity). In our implementation, we use the final lengths as the criteria: if, at the end of the simulation, the attacker's chain length is greater than or equal to the honest chain length, we consider the attack successful for that run. Each run's success or failure is recorded as a boolean flag. By running many trials (Monte Carlo simulation) with the same parameters, we can estimate the probability of a successful long-range attack under those conditions.

For example, we can run the simulation 100 times for an attacker stake fraction of 0.3 and initial difference $d = 5$ and observe how many of those trials the attacker catches up by the end. This gives an empirical success rate for that scenario.

By varying the parameters (attacker stake fraction, initial head-start d, number of slots), the implementation can explore different conditions. The results presented in the Results and Analysis chapter show how the probability of success of a long-range attack increases as the attacker's stake fraction grows and as the honest chain's head start shrinks. The outcome aligns with expectation: when the attacker controls a small stake (e.g., 10%) and the honest chain is significantly ahead, the attacker's fork almost never catches up (the process is like a random walk with a strong bias against the attacker). But if the attacker's stake is large or the honest lead is small, the attacker occasionally can reach parity. This implementation thus captures the essence of a PoS long-range attack in a simplified form, using probabilistic slot-by-slot simulation to model the race between an honest chain and an attacker's historically funded fork.

5.4 Selfish Mining Attack Simulation

The selfish mining scenario is implemented to analyse the well-known PoW attack where a miner (or a coalition of miners) withholds found blocks to gain a strategic advantage, as introduced by Eyal and Sirer [20]. In this attack, the adversary (selfish miner) does not immediately broadcast newly mined blocks. Instead, they keep a private chain and reveal blocks selectively to outperform honest miners and increase their share of the main chain's rewards. The implementation follows the state-machine strategy described in Eyal and Sirer's work, meaning the attacker's behaviour depends on the current lead (the length difference between the attacker's private blockchain and the public blockchain). This strategy determines when the attacker

reveals withheld blocks in order to achieve the highest probability of their blocks getting added to the main chain and thus earning the rewards, even with less than 50% of the total hash power.

5.4.1 Scenario Setup

The simulation for selfish mining models a blockchain growing over a sequence of blocks, with two groups of miners: the selfish miner (attacker) and the honest miners. The parameters of the model are the attacker's fraction of the total hash power and the honest miners' fraction.

We simulate block-by-block mining events where either the attacker or the honest network finds the next block. To begin, both the honest miners and the attacker start from a common genesis block at height 0 (so initially, the public chain and the attacker's private chain are in sync). Two separate chain representations are maintained: `public_chain` and `attacker_chain` (the attacker's secret fork chain). Both initially contain the genesis block.

A variable called `lead` exists to track the number of blocks by which the attacker's private chain leads the public chain. At genesis, `lead = 0` (the chains are equal). According to the selfish mining algorithm, as blocks are found, withheld, or published, `lead` may increase, decrease, or reset to zero.

5.4.2 Mining Process Simulation

The simulation then iterates, generating one new block at a time (as in a real blockchain where blocks arrive sequentially). In each iteration, either the attacker finds a block or the honest miners find a block. Which side finds the next block is determined probabilistically: the attacker finds the next block with probability (their hash power share), and the honest network finds it with probability. This is implemented by drawing a uniform random number and comparing it to. For example, if $\alpha = 0.3$, then 30% of the time, the attacker wins the mining race, and 70% of the time, an honest miner does. The core logic on each iteration is as follows:

- **If the attacker finds a block (with probability α):** The block is added to the attacker's private chain only and not revealed to the public. The attacker now has one more block than before on their fork, so the `lead` (attacker chain length minus public chain length) increases by 1. The public chain remains unaware of this block for now.

- **If an honest miner finds a block (with probability $1 - \alpha$):** The honest block is immediately added to the public chain (since honest miners follow the protocol and publish blocks). Now, the attacker must respond based on the current `lead`:

    1. **Case 1: Attacker lead = 0 (no private advantage)** – If the attacker had no unpublished blocks (their chain was not ahead), this honest block simply extends the public chain normally. The attacker's private chain, if it existed at all, is either empty or equal to the public chain; in either case, the attacker has no hidden blocks to compete with, so they accept that block. In the simulation, we append the honest block to the `public_chain`. The `lead` remains 0 (or becomes –1 and then reset to 0) because the attacker is not ahead; effectively, the attacker is now one block behind, but since the attacker's private fork is discarded or caught up, we treat it as no lead.

    2. **Case 2: Attacker lead = 1 (one block ahead)** – If the attacker had exactly one block in their private chain that was not published yet, and an honest miner finds a block, then the attacker immediately publishes their private block to prevent the honest block from getting too far ahead. By publishing, the attacker's previously secret block and the honest block are now competing at the same height. This results in a tie: there are two different blocks at the same height (one by the attacker, one by an honest miner). In Bitcoin, when two blocks are found at nearly the same time, the tie is resolved when a subsequent block is added to one of the chains, making it longer [10]. In the selfish mining strategy, when the attacker publishes their withheld block upon an honest block discovery in this scenario, the public network sees two branches of equal length. The honest block discovered is effectively orphaned if the attacker's block can gain the next block. In our simulation, we model this by adding the attacker's block to the public chain (so the public chain now reflects the attacker's block instead of the honest block, which we treat as overridden). The `lead` is reset to 0 because the attacker no longer has any unpublished blocks (they used their one block to tie).

3. **Case 3: Attacker lead ≥ 2 (two or more blocks ahead)** – If the attacker's private chain was at least two blocks ahead and then an honest miner finds a block, the attacker is far enough ahead to safely ignore the honest block. In this case, the attacker immediately publishes enough of their private blocks to still stay ahead by one block after publication. For example, if the attacker's `lead` was 2 and an honest block is found, the attacker publishes one of their two private blocks. This leaves the attacker still 1 block ahead (because now the public chain has grown by one attacker block, and the honest block is not adopted). If the `lead` was 3 and an honest block came, the attacker would publish two blocks, and so on. The simulation handles this by taking the attacker's private chain and releasing all but one of the hidden blocks to the public chain. After this, the attacker retains a `lead` of 1 block (one block remains secret). The honest block that was found is effectively ignored because the public chain has grown longer with the attacker's published blocks, so the honest block does not become part of the longest chain.

The following code snippet shows the above logic inside the simulation loop, focusing on how the attacker reacts when an honest block is found under different lead conditions:

Note: Boundary checks and implementation-specific details (e.g., total block limits) are omitted from the snippet for clarity.

```
# Randomly determine next block finder based on hash power fractions
if random.random() < (self.attacker_power / self.total_power):
    # Attacker finds a block: extend private chain only
    new_block = Block(attacker_length, miner=self.attacker_node)
    attacker_length += 1
    self.attacker_chain.append(new_block)
    lead += 1
else:
    # Honest miners find a block
    new_block = Block(public_length, miner=self.honest_node)
    if lead == 0:
        # No private lead: honest block extends the public chain
        self.public_chain.append(new_block)
```

```
        public_length += 1
    elif lead == 1:
        # Publish attacker's hidden block to tie
        self.public_chain.append(self.attacker_chain[-1])
        public_length += 1
        lead = 0
    else:
        # Attacker publishes enough blocks to remain exactly 1 block ahead
        blocks_to_publish = lead - 1
        self.public_chain.extend(self.attacker_chain[-blocks_to_publish:])
        public_length += blocks_to_publish
        lead = 1
```

Code Snippet 5.5: Selfish mining simulation logic for attacker's strategic block withholding and publishing

In this snippet, `attacker_hash_fraction` corresponds to α (the attacker's share of hash power). If the attacker finds the next block, we create a new `Block` (with height `attacker_height`) mined by the attacker and append it to the attacker's private chain, then increment both the private chain height and the `lead`. If an honest miner finds the block, we examine the `lead` value to decide the course of action. When `lead` $= 0$, the honest block is simply appended to the `public_chain` and the attacker's state remains with no lead. When `lead` $= 1$, the attacker's single hidden block (stored at the end of `attacker_chain`) is immediately published to the public chain (we append it to `public_chain`), resulting in a tie resolved in favour of the attacker's block (the honest block at that same height is ignored in our simulation). We then set `lead` $= 0$ because the attacker no longer has any unpublished blocks.

The code publishes all except one of the attacker's hidden blocks to the public chain when the `lead` ≥ 2 (ensuring the attacker stays one block ahead). We do this by taking the last `blocks_to_publish` $=$ `lead` $- 1$ blocks from the attacker's chain and appending them to the public chain, then updating the public chain height. The attacker retains one block privately (hence, the `lead` becomes 1).

In all cases, the honest block that triggered the response is not explicitly added to the public chain if the `lead` was 1 or more; effectively, the honest block gets orphaned because the

attacker's publication overshadows it. The simulation continues this process until a predetermined number of total blocks have been added to the public chain.

There is also some logic included at the end of the simulation to handle any remaining private blocks the attacker might still have when the loop ends. The system publishes all remaining private blocks to the public chain when the attacker completes the loop with a lead greater than 0. This ensures that the final public chain will contain as many attacker blocks as possible, assuming the attacker reveals everything once the mining race is over. After this, we count how many blocks in the public chain were mined by the attacker versus honest miners. The key metric gathered is the attacker's share of main chain blocks, which we compare against the attacker's hash power.

By running this simulation for various values of (attacker hash power fraction), we measure the outcome of selfish mining. According to theory, for certain thresholds (around 0.25–0.33), the attacker can obtain a larger portion of the blocks than their fair share by using the selfish mining strategy [21]. Our implementation collects the fraction of attacker-mined blocks on the main chain in each run and averages it over many runs for each. The results, plotted in the Results and Analysis chapter, show the attacker's actual block share vs. their hash power and can be compared to the theoretical curve from Eyal and Sirer's analysis. This validates that the simulation correctly reproduces the selfish mining advantage: for example, with around 0.3, the attacker's share of blocks in the longest chain may exceed 0.3 (meaning the attacker is earning more than 30% of the rewards, which is the incentive for the attack). At very high (close to 0.5), the attacker approaches controlling the majority of blocks; at very low, the attacker's strategy yields little to no advantage. These outcomes confirm that the implementation is consistent with known results and demonstrate the security implications: PoW systems are vulnerable to selfish mining when a miner has a substantial minority of the hash power, even though they may not have 50%.

To summarise, the selfish mining scenario follows a step-by-step simulation of block discovery and strategic publishing for its implementation. By following the known attack strategy [8][20] and updating the state of the public and private chains accordingly, the simulator captures how a selfish miner can exploit timing and information asymmetry to gain an outsized reward share. All key events (attacker finding a block, honest finding a block and attacker's response) are handled in code as per the described algorithm, and the data collected from these simulations provides insight into the effectiveness of the attack under different conditions.

5.5 Aggregate Analysis Scripts

Aggregate analysis scripts were implemented to efficiently evaluate each attack scenario across a range of parameter values. The scripts perform automated parameter sweeping through repeated simulations, which test different values of essential variables (such as the attacker's resource fraction). This automation is crucial for thoroughly exploring the attack's behaviour under different conditions and ensures that results are gathered consistently and reproducibly. By retrieving results across the entire parameter space, the aggregate scripts show how even just changing one parameter (e.g., the attacker's hash power in a selfish mining attack) influences the attack's success or impact.

Each aggregate script leverages Monte Carlo repetitions to achieve statistical reliability [9]. For every parameter setting being swept, the script executes the corresponding attack scenario numerous times (each with a different random seed). It computes summary statistics (such as the average outcome). Random fluctuations in the results are smoothed out by running the simulation multiple times, which yields more stable estimates of performance metrics and, therefore, increases confidence in the observed trends. To measure the attacker's block success rate, we can run 20 independent simulations for each attacker power level and then average the results. This makes sure that the reported mean is representative and not an anomaly of a single run.

Once the simulations are complete for all the parameter values, the data is aggregated into a tabular format and saved as a CSV file. Additionally, a graphical plot that visualises the relationship between the swept parameter and the attack outcome is generated automatically. These plots are saved to a folder (e.g., as PNG images) and serve as immediate visual summaries of the experiment. The script generates updated results charts automatically during each run without human intervention, providing the user with a quick way to analyse how the attack scales with different parameter inputs.

For illustration, a representative Selfish Mining aggregate script is shown in the code snippet below. The script allows the attacker's hash power ratio (α) to range between 5% and 90%. The simulation runs a set number of blocks for each α value before repeating this process multiple times (Monte Carlo trials) to calculate the mean fraction of blocks the attacker wins in the main chain. The calculated mean value gets compared against the attacker's fair share (equal to α, representing the baseline where the attacker only mines honestly). The code demonstrates how

the results are collected in a list of records, converted to a Pandas DataFrame [6], and then saved to a CSV file. A Matplotlib plot that shows the attacker's block share against their fair share across different α values is generated. And the script ends by finally saving the image.

Note: This snippet omits minor implementation details (e.g., file paths, directory creation) for clarity; the full simulation scripts contain these elements.

```
from simulation.attacks.selfish_mining import SelfishMiningScenario
import pandas as pd
import matplotlib.pyplot as plt

# Attacker's hash power fractions (  values) to evaluate
alphas = [0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.6, 0.7,
↪  0.8, 0.9]
blocks = 2000   # Number of blocks per simulation run
runs = 20       # Number of Monte Carlo repetitions per   value
results = []

# Run Selfish Mining simulations for each
for alpha in alphas:
    scenario = SelfishMiningScenario(alpha, 1 - alpha, total_blocks=blocks)
    scenario.run_monte_carlo(runs=runs)

    # Calculate average attacker fraction of blocks mined on main chain
    attacker_fraction_mean = sum(r['attacker_fraction'] for r in
    ↪  scenario.results) / runs

    # Record results (actual vs. fair share)
    results.append({
        'alpha': alpha,
        'attacker_fraction_mean': attacker_fraction_mean,
        'fair_share': alpha  # Fair share equals   (honest mining baseline)
    })

```

```
# Aggregate results into a DataFrame (file handling simplified for snippet clarity)
df = pd.DataFrame(results)
df.to_csv('selfish_mining_results.csv', index=False)

# Plot the attacker's actual block share versus their fair share
plt.plot(df['alpha'], df['attacker_fraction_mean'], 'bo-', label='Attacker Actual')
plt.plot(df['alpha'], df['fair_share'], 'r--', label='Fair Share')
plt.xlabel('Attacker Hash Power Fraction ( )')
plt.ylabel('Mean Attacker Fraction of Blocks')
plt.legend()
plt.savefig('selfish_mining_plot.png')
```

Code Snippet 5.6: Selfish Mining Aggregation Script

This Selfish Mining aggregate script is indicative of the approach used for other attacks in the project. Similar programs were implemented for the Sybil attack and the Long-Range attack, each tailored to sweep the relevant parameters (such as the number of Sybil identities or the stake fraction of an attacker) in those contexts. In all cases, the scripts automate extensive experiments, ensure systematic data collection, and produce both CSV datasets of results and corresponding graphical plots for visualisation. This comprehensive automation provides a reliable foundation for analysing and discussing the security implications of each attack scenario.

5.6 Summary

The implementation across all scenarios (Sybil attack, long-range attack, and selfish mining) focuses on modelling the essential mechanics of each attack in a simplified consensus setting. We isolate the consensus-level security behaviours by keeping the simulation abstract (omitting transaction details, network propagation delays, etc.). Each scenario's code was carefully validated against the conceptual descriptions in the literature (e.g., comparing with known results or formulas) to ensure correctness. Using short time-step iterations (rounds, slots, blocks) and probabilistic decisions allows us to observe emergent behaviour (such as an attack succeeding or

failing) under controlled parameters. The simulation results (attacker win rates, success flags, or block fractions) are used in the following chapters for evaluation and analysis.

Chapter 6

Results and Analysis

In this chapter, we present the experimental results for each attack scenario and analyse and evaluate how PoW and PoS blockchain systems fare against these vulnerabilities. The test setup is detailed for each section using the implemented simulation framework, and the outcomes are discussed. The final section provides an overall assessment of PoW and PoS resilience based on the obtained results.

6.1 Sybil Attack Analysis

This section evaluates how effective this resource-weighting is at neutralising a Sybil attacker, by comparing consensus outcomes with and without Sybil resistance.

6.1.1 Test Configuration

To quantitatively illustrate the impact of Sybil identities, we simulate a simple block production scenario with the following parameters:

- **Consensus Mode:** Both PoW and PoS were tested in separate runs (the Sybil mechanism is analogous in both, as both weight by resource).
- **Attacker Resource Fraction (α):** Varied from 0.1 to 0.9 (10% to 90% of total mining power or stake controlled by attacker).
- **Honest Nodes:** 5 honest identities (controlled by honest participants).

- **Attacker Sybil Nodes:** 20 identities all controlled by the single attacker (i.e., one adversary presents 20 Sybil identities).
- **Simulation Rounds:** 2,000 block-generation rounds per simulation run.
- **Monte Carlo Runs:** 10 runs for each α to average out randomness.
- **Metrics Recorded:** The fraction of blocks (out of 2,000) that the attacker wins under two conditions:
    1. *Weighted consensus* – where block selection probability is proportional to hash power or stake (PoW/PoS normal operation, Sybil-resistant)
    2. *Identity-based selection* – a hypothetical vulnerable scheme where each node (identity) has an equal chance regardless of resources (illustrating a Sybil attack scenario).

6.1.2 Results

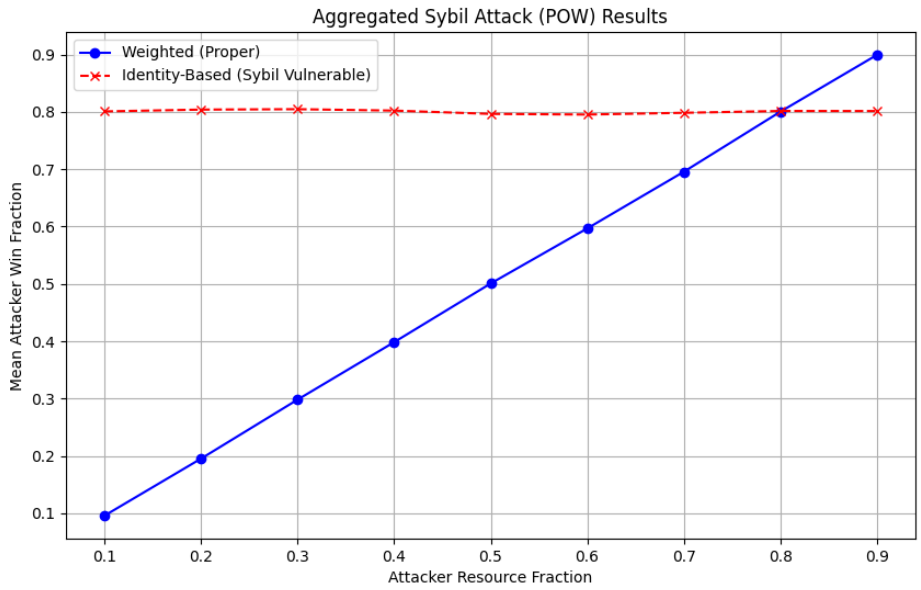

Figure 6.1: Aggregated PoW Sybil Attack Results

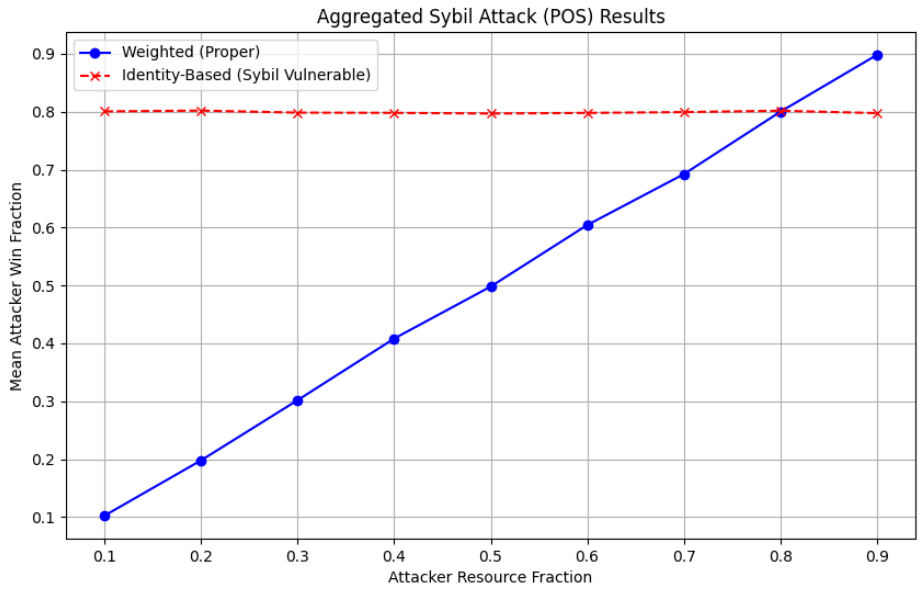

Figure 6.2: Aggregated PoS Sybil Attack Results

6.1.3 Results

The simulations demonstrate that resource-weighted consensus effectively limits Sybil power, while identity-based consensus proves to be severely vulnerable. The attacker's block production proportion in both PoW and PoS scenarios tracked their resource share (α). For instance, an attacker who possessed $\alpha = 0.1$ successfully controlled about 10% of blocks which showed proportional fairness.

The proportionate relationship between attackers and their resources continued to exist even when α grew, reflecting proper system functionality where adversaries cannot exceed their contributed resource proportion.

Contrastingly, identity-based consensus amplified attacker influence substantially due to Sybil nodes. An attacker with 20 out of 25 nodes (80% identities) consistently produced $\sim$ 80% of blocks, irrespective of their low actual resource fraction ($\alpha = 0.1$). The situation shows how small resource investment enables dominance through identity exploitation which reveals extensive Sybil attack vulnerability. Unlimited identity registration enables attackers who possess minimal resources to achieve complete dominance.

The simulation results showed identical outcomes between PoW and PoS because both consensus mechanisms effectively prevent Sybil attack amplification through resource-based weight-

ing. Sybil nodes alone do not enhance mining power or stake; influence increases strictly through costly resource acquisition. This supports the theoretical principle that both mechanisms achieve Sybil resistance by linking influence directly to resource contributions.

The simulations demonstrate that Nakamoto-style consensus possesses built-in Sybil resistance properties at the consensus level [16]. As long as honest participants maintain resource majority ($\alpha < 0.5$), attackers remain confined to proportional block production, consistent with protocol security assumptions. Without resource weighting, a trusted identity management system or centralisation would be required to mitigate Sybil vulnerabilities, as attackers could otherwise easily compromise the network.

The two mechanisms show equivalent Sybil resistance, yet Proof of Work may offer better network-level security because it requires Sybil miners to invest in physical hardware, while Proof of Stake allows easier creation of pseudonymous accounts. Nevertheless, splitting stake among multiple PoS accounts confers no advantage over a single aggregated stake account, reaffirming PoS's robust Sybil resistance through strict stake weighting.

The research confirms that both PoW and PoS systems operate correctly as secure systems because resource majority control prevents pure identity-based attacks.

6.2 Long-Range Attack Analysis (PoS)

This section evaluates Proof-of-Stake's vulnerability to long-range attacks by simulating scenarios where an attacker's historical stake fraction influences their ability to rewrite blockchain history.

6.2.1 Test Configuration

The long-range attack scenario was implemented for a PoS chain as follows:

Attacker Stake Fraction: Varied from 0.1 to 0.9 (10% to 90% of the total stake is controlled by the attacker's keys from genesis).

Honest Stake Fraction: The remainder of stake (i.e., 1 – attacker fraction) is held by honest participants. For example, if attacker stake = 0.4 (40%), honest stake = 0.6 (60%).

Initial Chain Discrepancy: 5 blocks. At the start of the simulation, the honest main chain is

assumed to be 5 blocks ahead of the attacker's alternative chain. This represents, for instance, that the honest network has a head start (or the attacker is trying to catch up from 5 blocks back in history).

Total Slots Simulated: 10,000 slots (block intervals) are simulated. Each slot, either an honest node or the attacker is chosen to create a block, proportional to their stake share.

Monte Carlo Runs: 20 runs for each stake fraction to estimate success probability.

Success Criterion: In each run, we check if the attacker-managed chain manages to catch up with and overtake the honest chain's length by the end of the 10,000 slots. The success rate (proportion of runs in which the attacker succeeds in overtaking) is recorded for each stake fraction.

6.2.2 Results

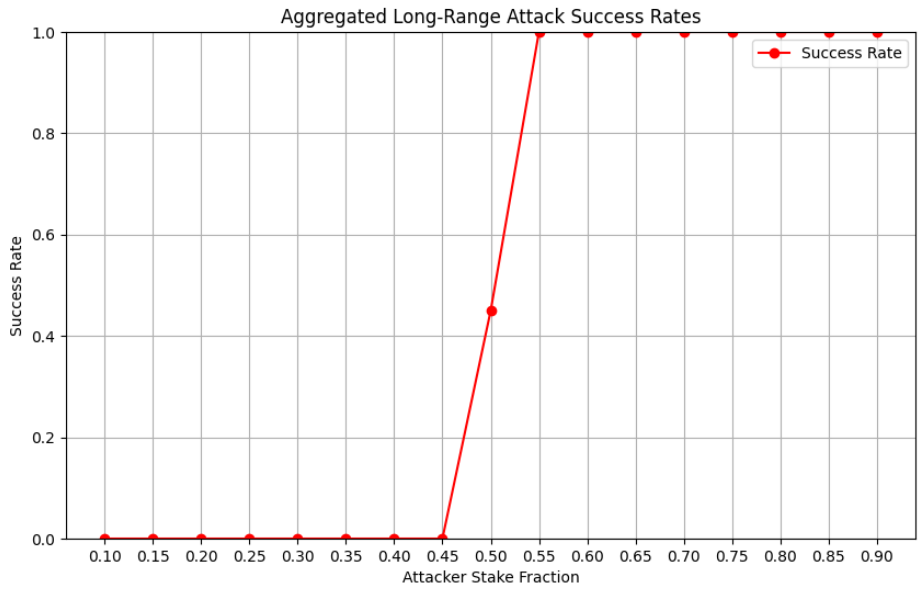

Figure 6.3: Aggregated Long-Range Attack Results

The long-range attack simulations demonstrate an apparent threshold of approximately 50-60% attacker stake. With a small stake fraction (e.g., $\alpha = 0.1$), the attacker failed to succeed in any trials, as expected, due to the extremely low probability of overcoming an honest majority and a 5-block lead. Even at $\alpha = 0.3$, success remained at zero. However, if the attacker's stake gets closer to equaling that of the honest stake, then the probability of success increases dramatically. At an even split (50% stake), the attacker succeeded in approximately 45% of trials, reflecting

a fair random walk scenario. At more than 50% stake ($\alpha = 0.55$), the attacker was successful in all the simulations within the $10,000$ slot simulations, as their block production probability was higher than that of honest nodes.

The table below demonstrates the results from the above experiment in detail:

attacker_stake_fraction	success_rate
0.1	0
0.15	0
0.2	0
0.25	0
0.3	0
0.35	0
0.4	0
0.45	0
0.5	0.45
0.55	1
0.6	1
0.65	1
0.7	1
0.75	1
0.8	1
0.85	1
0.9	1

The results illustrate that PoS, under standard assumptions of continuous honest majority participation, exhibits a security threshold analogous to PoW (50%). Below this threshold, historical rewrites are improbable; above it, attackers can freely rewrite history. However, critical differences exist between PoW and PoS attacks. PoW requires substantial ongoing computational and energy expenditure, making long-range historical revisions practically infeasible. Conversely, PoS attacks incur virtually no cost, enabling attackers to create extensive alternative histories offline.

The "nothing-at-stake" vulnerability in real-world PoS systems enables attackers to produce multiple forks simultaneously, which increases the chances of successful long-range forks [2]. The experimental model results failed to show that small stake majorities could successfully

take over the honest chain in every simulation run. The attacker achieves success according to simulation logic when their chain reaches or exceeds the length of the honest chain during the specified time slot period. The simulations produced consistently close to zero success probabilities for attackers who maintained minority stake positions because block creation probabilities directly related to stake proportions. The results demonstrate how basic models fail to represent historical security weaknesses in PoS systems accurately.

Our analysis confirms PoS's necessity for additional security mechanisms due to its intrinsic vulnerabilities in long-range attacks, especially when honest participation declines or stakeholder keys become compromised. Unlike PoW, whose cumulative computational cost inherently secures historical states, PoS systems must actively manage vulnerabilities to maintain long-term security integrity.

6.3 Selfish Mining Attack Analysis (PoW)

This section analyses the impact of selfish mining in PoW blockchains by simulating how withholding and strategically releasing blocks allows attackers to disproportionately increase their rewards relative to their hash power.

6.3.1 Test Configuration

The selfish mining scenario was simulated using a discrete event model of blockchain growth. Key parameters and steps were:

Attacker Hash Power (α): Varied over a range from 0.05 up to 0.9 (5% to 90% of total mining power controlled by the selfish miner).

Honest Hash Power: $1 - \alpha$ (the rest of the mining power is held by honest miners who follow the protocol normally).

Simulation Length: 2,000 blocks were mined in each simulation run (sufficient to observe steady-state block share outcomes).

Strategy Model: The attacker follows the state-machine strategy from Eyal and Sirer's paper [20]. Specifically, the attacker maintains a private chain and a lead counter. They withhold blocks when advantageous and publish them to race ahead of the public chain according to the rules: e.g. if the attacker is one block ahead and the honest network finds a block (leading to a

tie), the attacker immediately publishes their hidden block to regain advantage; if the attacker gets two blocks ahead, they keep mining privately to extend their lead, and so on (see Code Snippet 2 for full strategy). Honest miners always mine on the longest chain they know of (or randomly choose one in case of a tie).

Monte Carlo Runs: 20 independent runs for each α to average out variance due to the random nature of block discovery and tie-breaking.

Metrics Recorded: The primary metric is attacker's fraction of main chain blocks (i.e. what portion of the 2000 published blocks were mined by the attacker) averaged over the runs. We compare this to the attacker's hash power fraction (α) which represents their expected share if they mined honestly. Any excess of the attacker's block fraction over α indicates a successful exploit of the strategy.

6.3.2 Results

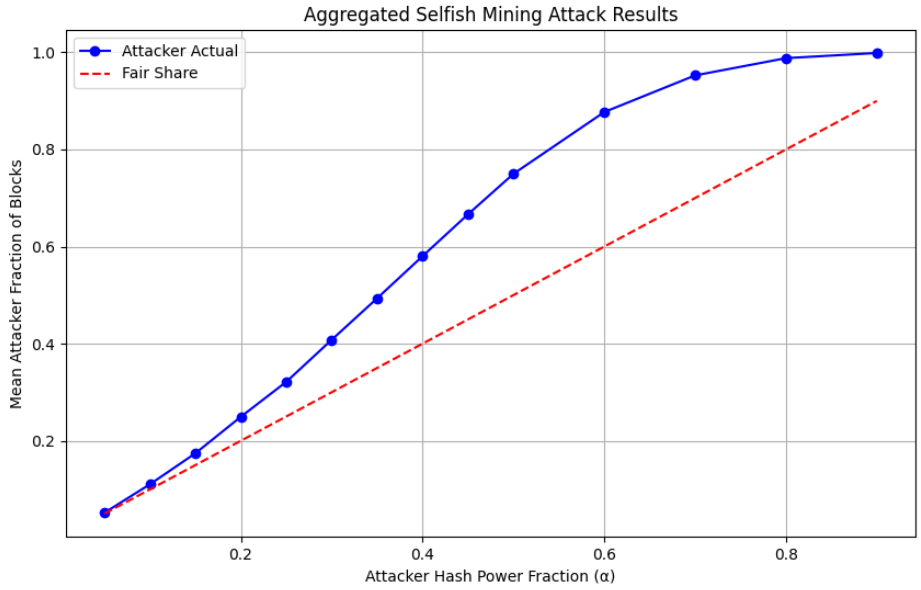

Figure 6.4: Aggregated Selfish Mining Attack Results

alpha	attacker_fraction_mean	fair_share
0.05	0.052025	0.05
0.1	0.1105	0.1
0.15	0.174275	0.15
0.2	0.249475	0.2
0.25	0.3214	0.25
0.3	0.408025	0.3
0.35	0.493525	0.35
0.4	0.58025	0.4
0.45	0.66645	0.45
0.5	0.749775	0.5
0.6	0.8771	0.6
0.7	0.952575	0.7
0.8	0.98815	0.8
0.9	0.998775	0.9

Simulation results confirm a threshold in hash power at which selfish mining yields disproportionate rewards. At low hash power levels ($\alpha = 0.05$), attackers obtained approximately 5.2% of blocks, slightly exceeding their proportional share, primarily due to orphaned secret blocks. At $\alpha = 0.10$, attackers secured around 11.05%, closely matching their proportional stake.

As hash power approaches 30% ($\alpha = 0.3$), attackers gain a significant advantage, capturing roughly 40.8% of blocks, marking a clear deviation from proportional fairness. The simulations show that there is a significant advantage at $\alpha = 0.35$, and the attackers got approximately 49.35% of blocks, substantially exceeding their proportional share.

At higher attacker hash powers, selfish mining advantages become even more pronounced. Attackers secured approximately 58% of blocks at $\alpha = 0.4$ and 75% at $\alpha = 0.5$. With further increases in hash power ($\alpha = 0.6$), attackers captured approximately 87.7% of blocks, proving their capability to dominate block production significantly below the conventional 51% majority threshold. This substantial advantage incentivises rational miners to join selfish mining pools, which may lead to network centralisation. Historically, concerns regarding Bitcoin mining pools approaching 30–40% hash power reflect the significant risks highlighted by these results, which go beyond mere double-spend attacks to fundamental issues of incentive compatibility.

The simulation results are consistent with the previous works (Eyal and Sirer, 2014): The profitability thresholds were reported to be around $\alpha \approx 0.25$ (ideal) and $\alpha \approx 0.33$ (realistic), and the subsequent studies have shown even lower thresholds under the specific conditions ($\sim 23\%$).

In summary, the analysis demonstrates a critical PoW vulnerability where minority attackers are able to achieve disproportionate control.

6.4 Edge-Case Testing

To verify that the simulator behaves correctly in extreme conditions, we conducted additional runs for each attack with boundary input values:

6.4.1 Test Configuration

- **Long-Range Attack (PoS)** – The attacker's historical stake fraction at the fork was set to $attack\_frac = 0.1, 0.495$, and 0.9. And we vary the initial lead between 3 and 20 to test both minimal and significant head-starts.

- **Selfish Mining (PoW)** – The attacker's hash power share was set to $\alpha = 0, 0.33$, and 0.6. These cases represent no attacker (baseline), the theoretical $\sim 33\%$ threshold where selfish mining becomes profitable, and a majority attacker scenario. All other network conditions (e.g., block interval, tie-breaking) remained as in earlier experiments.

- **Sybil Attack** – An extreme scenario was tested where the attacker controls 100 identities but 0% of the total stake or hash power. We evaluate this under two consensus modes: (a) the normal resource-weighted protocol (stake or hash weighted) and (b) a naive protocol where each node has an equal vote regardless of stake. Other parameters (number of honest nodes, total stake) mirror the earlier Sybil setup.

6.4.2 Long-Range Attack Results

Table below acts as a key to the graph results shown in this section:

Test ID	attack_frac	initial_diff
Test 1	0.01	3
Test 2	0.1	20
Test 3	0.495	3
Test 4	0.495	20
Test 5	0.9	3
Test 6	0.9	20

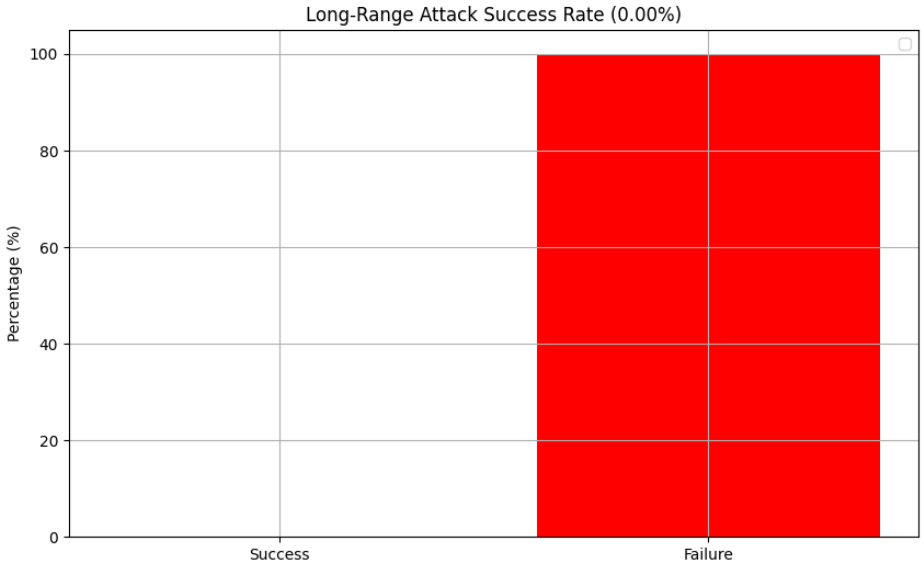

Figure 6.5: Test 1

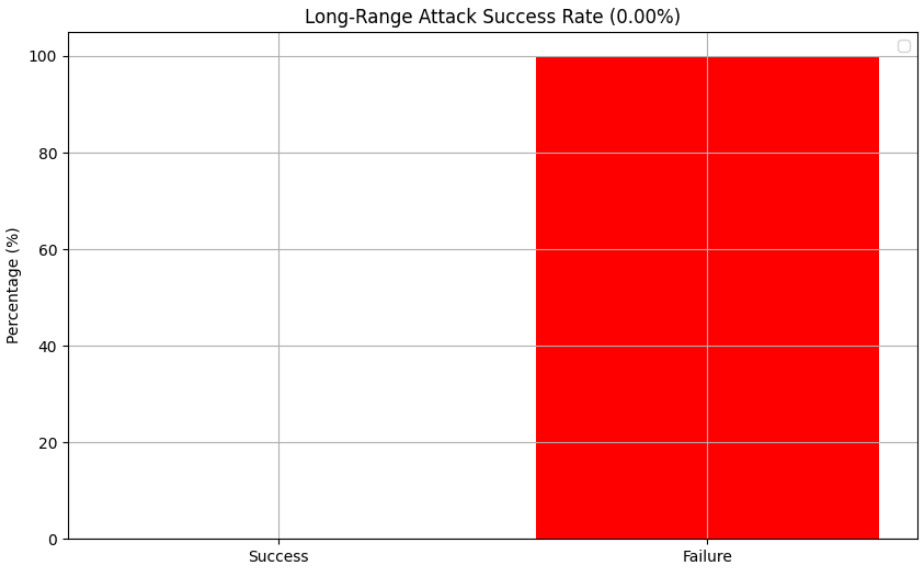

Figure 6.6: Test 2

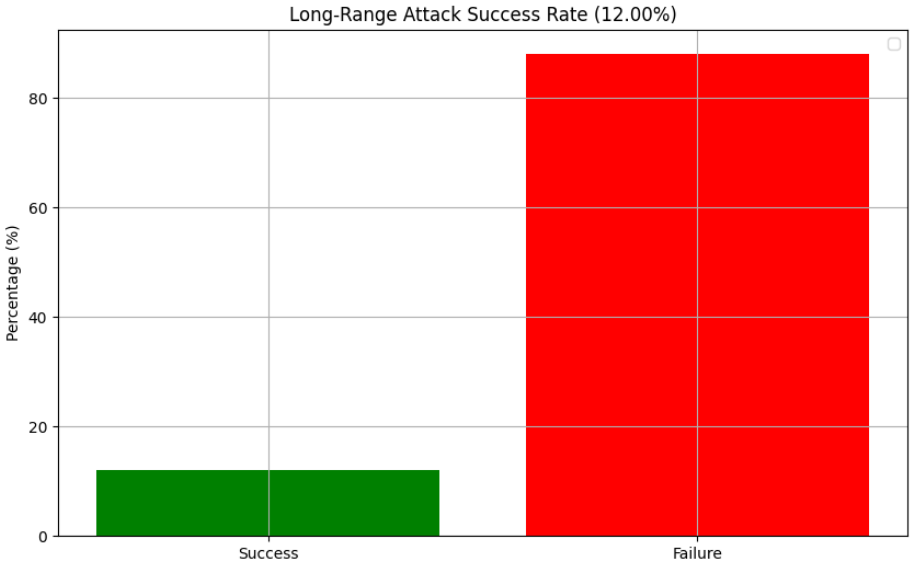

Figure 6.7: Test 3

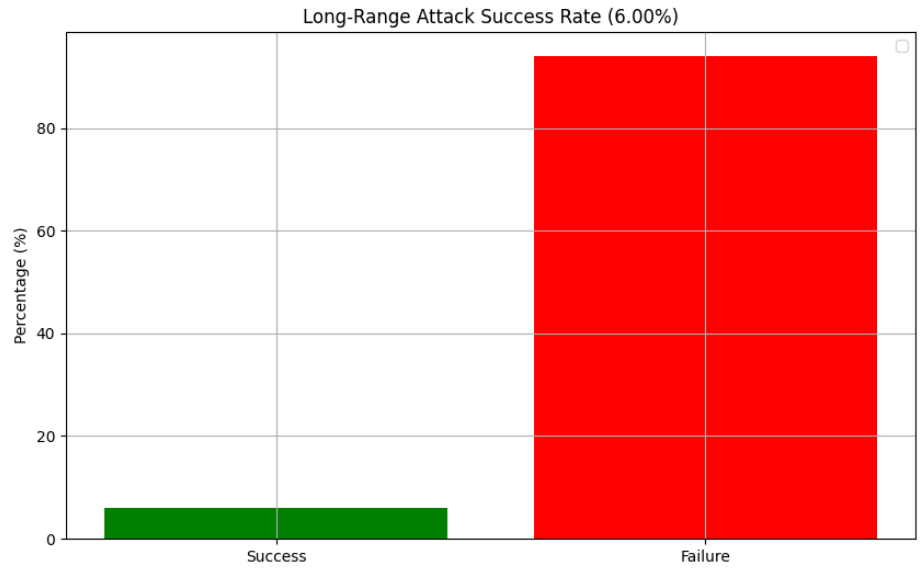

Figure 6.8: Test 4

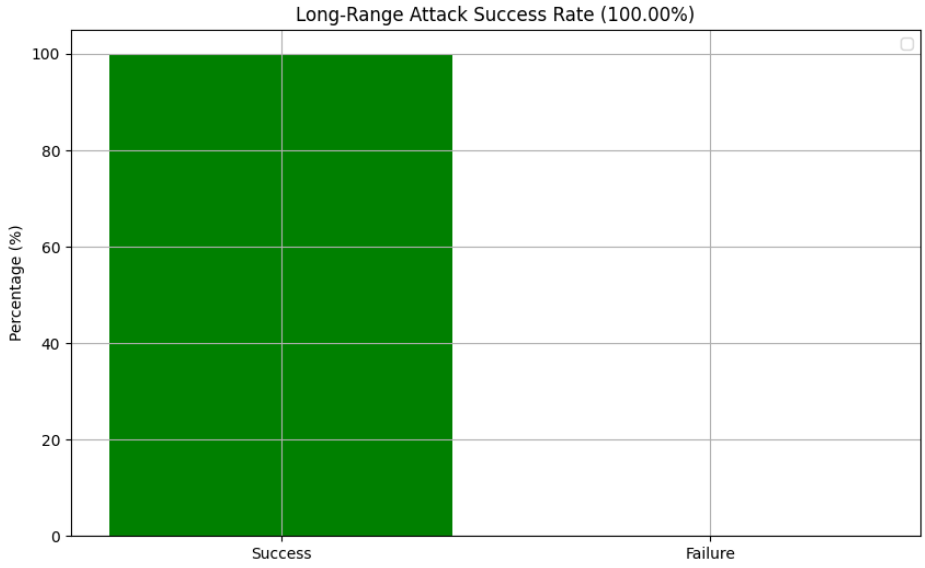

Figure 6.9: Test 5

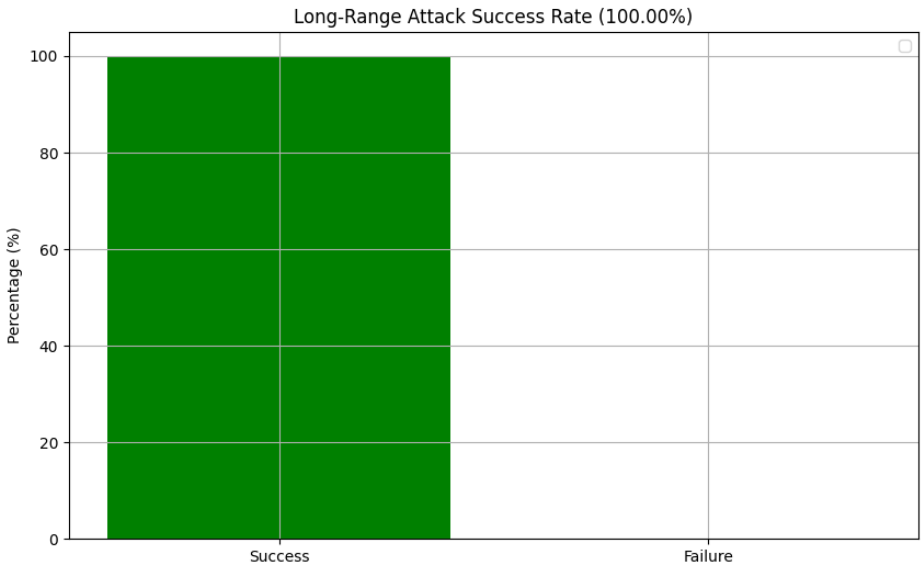

Figure 6.10: Test 6

6.4.3 Selfish Mining Attack Results

Table below acts as a key to the graph results shown in this section:

Test ID	alpha
Test 1	0
Test 2	0.33
Test 3	0.6

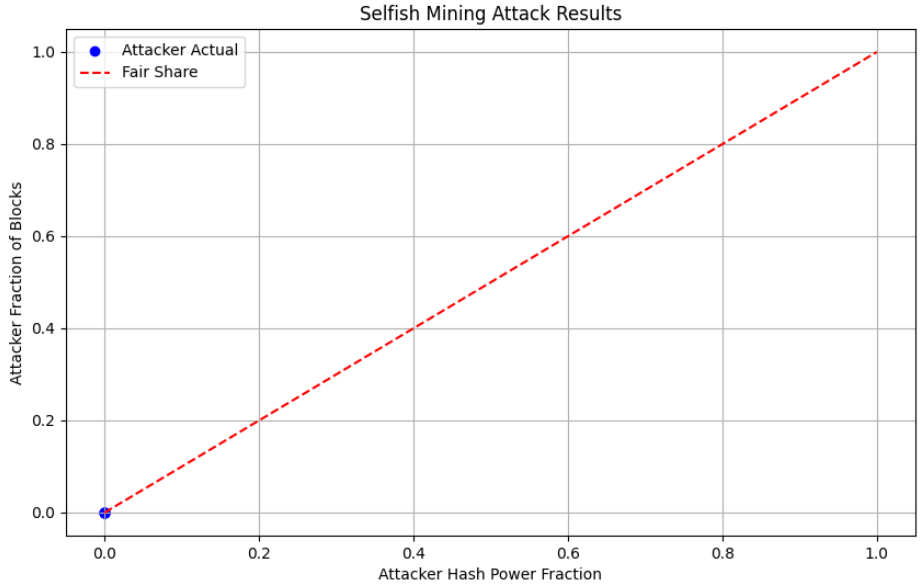

Figure 6.11: Test 1

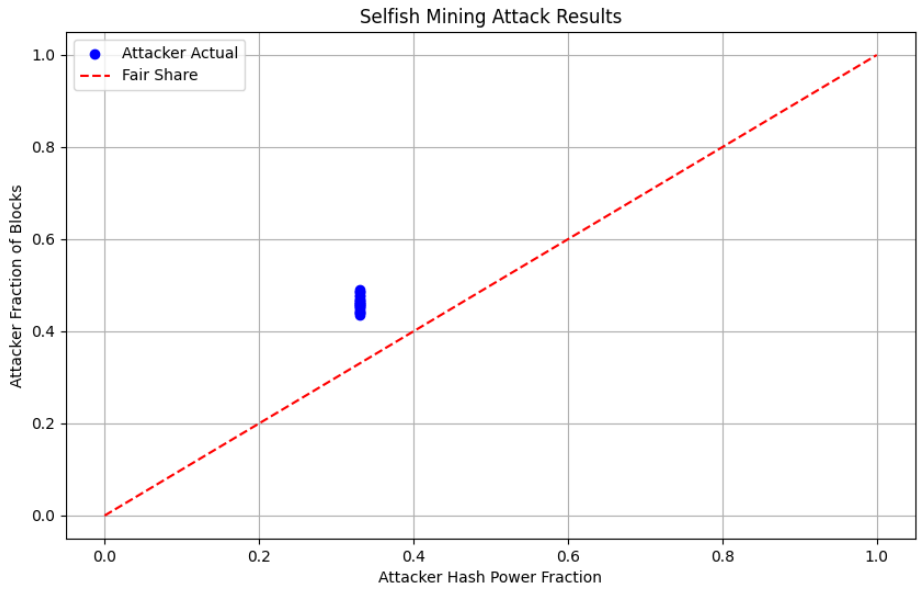

Figure 6.12: Test 2

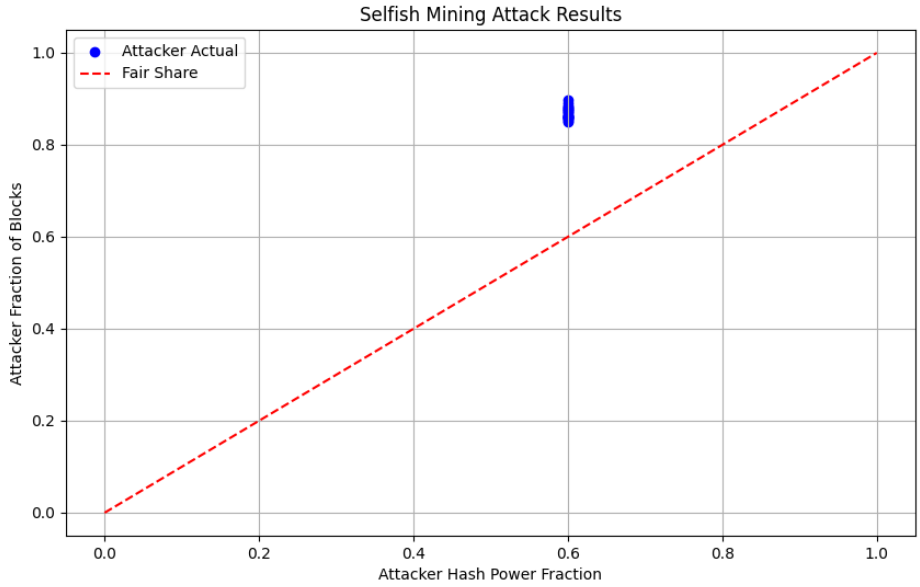

Figure 6.13: Test 3

6.4.4 Sybil Attack Results

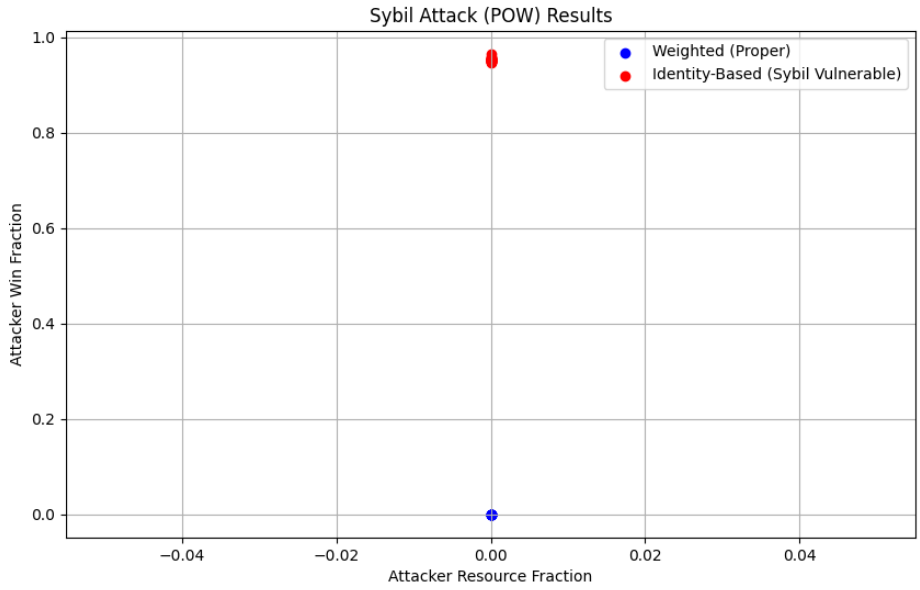

Figure 6.14: Test 1

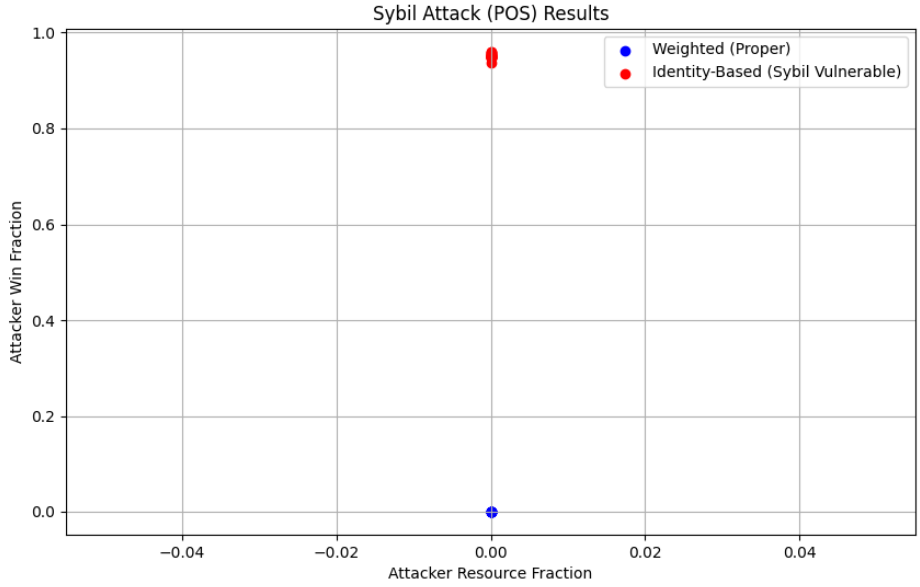

Figure 6.15: Test 2

6.4.5 Results Summary

The results from edge case tests confirmed the simulation predictions based on consensus theory.

In long-range attack scenarios, an attacker with attack_frac = 0.1 stake never surpassed the honest chain due to insufficient stake. At attack_frac = 0.495 stake, which is just slightly below the majority, attackers occasionally overcame minor honest chain leads (few blocks) after extended periods, while larger initial deficits prevented takeovers much further, as seen in Figure 6.8. Conversely, attackers who possessed a 0.9 attack_frac stake could rewrite historical data whilst quickly surpassing moderate honest chain leads, confirming that reliable takeovers require greater than 50% stake.

In selfish mining scenarios, attackers at $\alpha = 0$ stake had no effect. At $\alpha = 0.33$, attackers' long-term block slightly exceeded their hash rate (33%), reflecting the selfish mining strategy's profit point. With $\alpha = 0.6$, attackers disproportionately dominated block rewards (greater than 60%), frequently causing honest blocks to be orphaned. This demonstrates that attackers who possess more than half of the hash power can consistently gain control over the blockchain while obtaining excessive rewards.

During Sybil attacks, resource-weighted consensus rendered attackers with 0% stake and multiple identities (100 nodes) ineffective, as honest nodes monopolised mining power. This confirms that resource-based PoW/PoS systems resist attackers lacking stake or work contributions. Conversely, a naive equal-node scheme enabled attackers' numerous identities to dominate block selections despite zero stake, highlighting the severe vulnerability to classic Sybil attacks when identity creation has no associated cost.

6.5 Comparative Evaluation of PoW vs PoS Resilience

6.5.1 Sybil Attacks

The simulation results show that PoW and PoS both demonstrate effective Sybil resistance. According to their design principles, the fundamental resource acquisition of computational power or economic stake determines substantial influence in either system. Initially motivated by Bitcoin's PoW design, PoS similarly substitutes stake for hash power. The designs of both systems create equal resource acquisition costs, which prevent either from offering superior Sybil resistance. The physical barriers of PoW are based on hardware and energy consumption, in

contrast with PoS economic barriers related to monetary concentration. However, both systems need substantial investment to launch attacks.

6.5.2 Long-Range Attacks

A key divergence emerges here. PoW inherently resists long-range attacks; attackers cannot retroactively create a superior alternative chain without sustained majority hash power since the genesis block is due to the cumulative difficulty barrier. Conversely, PoS remains vulnerable to long-range attacks by simulating alternative histories with past stakeholders' keys.

Analysis indicates that attackers with a sub-50% stake rarely succeed if honest stakeholders are active. However, PoS risks escalate significantly if the honest stake becomes inactive, enabling even minor attackers to succeed. Therefore, PoS implementations require supplementary protections (e.g., checkpoints, slashing mechanisms, network assumptions) to match PoW's intrinsic historical immutability.

6.5.3 Selfish Mining

Here, advantages invert. PoW shows weaknesses in selfish mining when a miner or pool controls more than approximately 33% of the total hash power. The simulations show attackers can obtain more blocks and rewards than their fair share, undermining blockchain fairness and security. This strategic, subtle exploitation relies on competitive block production absent in studied PoS protocols (e.g., Ouroboros, Casper) [23][15], which utilise preselection or stake-weighted randomness. Consequently, PoS inherently resists selfish mining, whereas PoW demands vigilant monitoring and protocol adjustments.

6.6 Functional Requirements Evaluation

This section evaluates the implemented system and research performed against each functional requirement outlined in Chapter 3.2. A brief supporting description is included to clearly indicate the extent to which each functional requirement has been achieved.

ID	Requirement Name	Supporting Evidence
FR1	PoW and PoS models	Clearly implemented distinct PoW and PoS consensus mechanisms in `engine.py`, encapsulating key logic differences.
FR2	Sybil Attack Scenario	Implemented in `sybil.py` with accurate comparison of resource-weighted and identity-based voting methods.
FR3	Long-Range Attack Scenario	Comprehensive probabilistic race scenario implemented in `long_range.py`, clearly matching design in Chapter 4.
FR4	Selfish Mining Attack Scenario	Faithful reproduction of Eyal and Sirer's selfish mining strategy in `selfish_mining.py`, results closely align with theoretical outcomes (see Chapter 6.3).
FR5	Parameterisation	Flexible parameter adjustments (e.g., attacker fraction, chain gaps) consistently supported through scenario initialisation parameters.
FR6	Utilise Multi-run Monte Carlo Simulations	Multi-run execution logic and statistical aggregation successfully demonstrated in all aggregate scripts (e.g., `aggregate_selfish.py`).
FR7	Results Logging	All simulations systematically log results into CSV format for easy analysis and verification, as described in Chapter 5.5.
FR8	Results Visualisation	Implemented concise and informative matplotlib visualisations clearly illustrating simulation outcomes (e.g., Figures 6.1-6.15).

6.7 Non-Functional Requirements Evaluation

This section evaluates the implemented system and research performed against each non-functional requirement outlined in Chapter 3.3. A brief supporting description is included to clearly indicate the extent to which each functional requirement has been achieved.

ID	**Requirement Name**	**Supporting Evidence**
NFR1	Accuracy and Validity	Simulation outcomes consistently match theoretical expectations (Chapters 6.1–6.4), validating both the accuracy and robustness of the implemented models.
NFR2	Efficiency	Capable of efficiently running thousands of simulation iterations with acceptable runtime performance (e.g., 2000 blocks per run as in Chapter 6.3).
NFR3	Modularity and Clarity	The modular structure with clearly separated responsibilities across Python scripts (`engine.py`, scenario scripts, aggregate scripts) significantly enhances maintainability.
NFR4	Reproducibility	Configurable random seeds ensure full reproducibility, effectively demonstrated through consistent results across multiple identical runs.
NFR5	Usability for Experimentation	CLI parameter interface designed for straightforward execution and easy experimentation (validated extensively during practical testing).
NFR6	Data Integrity	Results consistently and accurately reflect simulation events, confirmed through comprehensive cross-verification of logs and visual outputs.

Chapter 7

Legal, Social, Ethical and Professional Issues

7.1 Legal Issues

The simulation developed operates purely using synthetic data, with no personal user information involved, so GDPR regulations do not apply [3]. All the data used throughout the project for testing purposes was artificially generated, so no actual user data was collected or analysed.

Furthermore, the simulation is run in a closed simulated environment and has no connection with actual cryptocurrency systems or assets. Therefore, the work falls outside the financial regulatory scope, as it involves no real monetary value or financial service.

7.2 Social and Ethical Issues

The simulator provides an unbiased representation of honest participants and adversaries in the blockchain network, which is important for security evaluation. The experimental setup and analysis explicitly stated all assumptions and limitations to preserve ethical integrity and achieve transparency in the results. The research maintains academic standards by evaluating Sybil and long-range and selfish mining attacks by using established attack definitions and methodologies that prevent unethical behaviour [20][19][13]. The research did not result in any real-world system damage because all attack simulations were conducted to boost blockchain

security.

7.3 Sustainability

The Proof of Work component of the simulation replicates the computation-intensive aspects of mining, but it does not perform actual hashing at a scale that would consume significant energy. While the simulator models PoW mining processes, it does not perform actual block mining on a real network or burn electricity on cryptographic puzzles. Thus, the environmental footprint of the simulation is minimal. The study contrasts PoW with PoS partly to highlight environmental implications: Blockchains which use PoW require substantial energy use by design, whereas PoS systems eliminate the need for wasteful mining competition. Our results show that PoS has dramatically lower resource usage, which aligns with real-world observations that PoS reduces energy consumption by orders of magnitude [13]. PoS was included for security comparison and to emphasise its improved sustainability. Demonstrating that similar security can be achieved with far less energy (for example, Ethereum's move from PoW to PoS cut its energy use by 99% [13]) underscores the environmental advantages of PoS in blockchain design.

7.4 Professional Standards

The research followed the BCS Code of Conduct [1] and IEEE Code of Ethics [4] during its development and analysis phase. The work applied essential principles from these professional standards throughout the research:

Public Interest and Avoidance of Harm: The research serves the public interest by aiming to improve blockchain security. It protected users by confining all tests to a virtual environment with no impact on real systems or users.

Integrity and Data Honesty: All results from the simulation experiments are reported honestly, without fabrication or misrepresentation. The report contains documented assumptions and limitations which maintain both data integrity and transparency.

Professional Competence: The work demonstrated professional competence through careful execution of established research methods and best practices in accordance with BCS and IEEE standards. **Accountability and Transparency:** The experimental process was recorded

thoroughly for replication and peer review, and the code was developed responsibly with proper documentation. All sources used in algorithm development were properly cited to credit the original authors, maintaining academic honesty.

By adhering to these standards, the project ensured ethically sound conduct and reliable results, consistent with professional computing practices.

7.5 Broader Impact

This project focuses on simulation-based analysis of PoW and PoS consensus mechanisms, and how they react under defined attack conditions. Although it does not propose new protocols or directly interact with real-world networks, it contributes to understanding how classic attack strategies behave under varied resource distributions. The research confirms that PoW and PoS need extra design elements to defend against sub-majority attacks such as selfish mining and long-range attacks. Resource-based Sybil resistance continues to be a fundamental advantage of both systems.

While not directly impacting live blockchains, the simulation output may support future studies on improving consensus robustness. For example, the findings about the selfish mining threshold at approximately 30% hash power and the requirement for PoS checkpointing mechanisms match existing observations in academic and industry research [20]. The project also indirectly promotes public trust by clarifying attack vectors and emphasising the need for continued protocol hardening. Ultimately, the broader relevance lies in contributing empirical evidence for comparing resilience and trade-offs in consensus system design.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

This dissertation compared Proof of Work (PoW) and Proof of Stake (PoS) under Sybil attacks, long-range attacks, and selfish mining. Both PoW and PoS fundamentally serve as Sybil resistance mechanisms by requiring a costly resource for participation (computing power or staked coins) [27]. This makes it infeasible for an attacker to gain voting power simply by creating many identities. However, if an attacker does acquire a majority of the hashing power or stake (a 51% attack), they can override the consensus in either system – a common vulnerability once the Sybil barrier is surpassed.

The analysis of selfish mining reveals a sharp contrast between PoW and PoS. In PoW, a colluding miner with far less than 51% hash power can exploit the protocol: by withholding and privately mining blocks, they extend a secret fork and gain more than their fair share of rewards. Eyal and Sirer showed that a pool with ∼30% of total hash power can successfully execute this attack [20]. Our simulations confirm that such a miner (around one-third of the hash rate) can earn disproportionately high rewards, undermining the fairness of PoW mining. In contrast, PoS protocols can significantly reduce selfish mining through careful design. The block creation process in PoS operates through random assignments or scheduled rotations which restrict validators from building a private lead. Furthermore, penalties within PoS systems known as stake slashing penalise validators who try to equivocate or fork the chain which eliminates their motivation for selfish actions. Indeed, the Ouroboros PoS protocol is

designed such that honest behaviour is a Nash equilibrium, neutralising any benefit from selfish mining [23]. Thus, with appropriate mechanisms, PoS can resist the reward-based attack that threatens PoW, albeit by adding protocol complexity.

Long-range attacks are a vulnerability unique to PoS. In PoW, rewriting the transaction history becomes practically impossible without majority hash power due to the cumulative proof-of-work required. In PoS, by contrast, an attacker who obtains the private keys of old validators or a large stake from the past could forge an alternative long-term chain history at negligible cost. Without safeguards, a new node might not be able to distinguish such a malicious fork from the legitimate chain since both satisfy the basic consensus rules (this is the *nothing-at-stake* problem). To counter this, PoS networks introduce additional measures and trust assumptions. One solution is to use weak subjectivity checkpoints: nodes are required to trust recent checkpoints or social consensus inputs that declare the true chain head, thereby invalidating any fork that diverged long ago [14]. In essence, PoS adds a layer of governance or social coordination to cement the ledger's history, trading off some decentralised trustlessness to eliminate long-range forks [14]. This trade-off is absent in PoW, which offers a constantly synced chain from genesis backed solely by computational work – but at the cost of energy-intensive operation and slower finality.

To conclude, PoW offers proven security at the cost of high energy usage and potential miner centralisation [20]. At the same time, PoS provides efficiency and adaptability but relies on more complex trust and incentive mechanisms [14]. Ultimately, one paradigm has no strict superiority over another; each presents distinct trade-offs, so the optimal choice depends on the system's priorities.

8.2 Future Work

Future work could extend this project by enhancing the simulation models with more realistic network conditions and participant behaviour. For example, incorporating network latency and fluctuations in mining power or stake distribution would provide a more accurate reflection of real-world conditions. This refinement would help determine whether results like the selfish mining threshold observed under idealised assumptions hold true once such practical factors are considered. In addition, the PoS simulation can integrate defence mechanisms such as checkpointing and slashing to evaluate their effectiveness in mitigating long-range attacks.

Further work could involve controlled experiments on private testnets or forks of existing blockchain networks to validate the findings under live conditions. This will demonstrate how actual network participants and protocols respond to attempted attacks in practice. Lastly, analysing empirical data from operational PoW and PoS networks would refine the existing attack models and validate the identified vulnerability thresholds against observed behaviours, ultimately increasing the applicability and robustness of the presented findings.

References

[1] Bcs code of conduct. https://www.bcs.org/media/2211/bcs-code-of-conduct.pdf. [Accessed 02-04-2025].

[2] Ethereum 2.0's nothing at stake problem. https://blogs.cornell.edu/info2040/2018/11/29/ethereum-2-0s-nothing-at-stake-problem/. [Accessed 29-03-2025].

[3] General data protection regulation (gdpr) – legal text. https://gdpr-info.eu/. [Accessed 05-04-2025].

[4] Ieee code of ethics. https://www.ieee.org/about/corporate/governance/p7-8.html. [Accessed 02-04-2025].

[5] Monte carlo simulation: What it is, how it works, history, 4 key steps. https://www.investopedia.com/terms/m/montecarlosimulation.asp. [Accessed 01-04-2025].

[6] pandas.dataframe — pandas 2.2.3 documentation. https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html. [Accessed 30-03-2025].

[7] Plotting in python: Comparing the options. https://anvil.works/blog/plotting-in-python. [Accessed 20-03-2025].

[8] Statistical detection of selfish mining in proof-of-work blockchain systems. https://www.nature.com/articles/s41598-024-55348-3. [Accessed 26-03-2025].

[9] System reliability analysis and diagnosis by monte carlo method. https://www.sciencedirect.com/science/article/pii/S1474667017661263. [Accessed 01-04-2025].

[10] What happens when two blocks are mined simultaneously? bitcoin chain splits explained.

https://braiins.com/blog/bitcoin-chain-splits-explained-temporary-forks. [Accessed 26-03-2025].

[11] What is a sybil attack: Examples & prevention. https://www.imperva.com/learn/application-security/sybil-attack/. [Accessed 25-03-2025].

[12] Foteini Baldimtsi, Varun Madathil, Alessandra Scafuro, and Linfeng Zhou. Anonymous lottery in the proof-of-stake setting. In *IEEE 33rd Computer Security Foundations Symposium (CSF)*, pages 318–333. IEEE, 2020.

[13] Carl Beekhuizen. Ethereum's energy usage will soon decrease by 99.95%. *Ethereum Foundation Blog*, 2021.

[14] Vitalik Buterin. Proof of stake: How i learned to love weak subjectivity. *Ethereum Blog*, 2014.

[15] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *arXiv preprint arXiv:1710.09437*, 2017.

[16] Archie Chaudhury. What is nakamoto consensus and how does it power bitcoin? https://bitcoinmagazine.com/guides/what-is-nakamoto-consensus-bitcoin. [Accessed 02-04-2025].

[17] RocketMe Up Cybersecurity. What is a 51% attack? https://medium.com/@RocketMeUpCybersecurity/51-attacks-how-they-work-ramifications-safeguards-8192de3d6162, Nov 2024. [Accessed 01-04-2025].

[18] Evangelos Deirmentzoglou, Georgios Papakyriakopoulos, and Constantinos Patsakis. A survey on long-range attacks for proof of stake protocols. *IEEE Access*, 7:28712–28725, 2019.

[19] John R. Douceur. The sybil attack. In *International Workshop on Peer-to-Peer Systems*, pages 251–260. Springer, 2002.

[20] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. *Communications of the ACM*, 61(7):95–102, 2018.

[21] Colin Finkbeiner, Mohamed E Najd, Julia Guskind, and Ghada Almashaqbeh. Sok: Time

to be selfless?! demystifying the landscape of selfish mining strategies and models. *Cryptology ePrint Archive*, 2025.

[22] Abhishek Guru, Bhabendu Kumar Mohanta, Hitesh Mohapatra, Fadi Al-Turjman, Chadi Altrjman, and Arvind Yadav. A survey on consensus protocols and attacks on blockchain technology. *Applied Sciences*, 13(4):2604, 2023.

[23] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual International Cryptology Conference*, pages 357–388. Springer, 2017.

[24] Sunny King and Scott Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. *Self-published*, 2012.

[25] Stefanos Leonardos, Daniël Reijsbergen, and Georgios Piliouras. Weighted voting on the blockchain: Improving consensus in proof of stake protocols. *International Journal of Network Management*, 30(5):e2093, 2020.

[26] Nadisha Madhushanie, Sugandima Vidanagamachchi, and Nalin Arachchilage. Selfish mining attack in blockchain: A systematic literature review. *International Journal of Information Security*, 23(3):2333–2351, 2024.

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

[28] Moritz Platt and Peter McBurney. Sybil in the haystack: A comprehensive review of blockchain consensus mechanisms in search of strong sybil attack resistance. *Algorithms*, 16(1):34, 2023.

[29] Yanbin Sun, Zhihong Tian, Yuhang Wang, Mohan Li, Shen Su, Xianzhi Wang, and Dunqiu Fan. Lightweight anonymous geometric routing for internet of things. *IEEE Access*, 7:29754–29762, 2019.

Appendix A

Appendix

A.1 User Guide

This user guide explains how to run the simulation framework, both for individual scenarios and for the aggregate scripts, to generate results in CSV format along with plotted graphs.

Running the Executable

Open a terminal and navigate to the folder containing `main.exe`, then use the commands below.

Run a Single Simulation

Sybil Attack (PoW or PoS):

```
main.exe sybil
```

Code Snippet A.1: Run default Sybil attack

```
main.exe sybil --consensus pos --attack_frac 0.3 --honest_nodes 5
↪  --sybil_nodes 20 --rounds 1000 --runs 10
```

Code Snippet A.2: Customised Sybil attack (PoS)

Selfish Mining (PoW):

```
main.exe selfish
```

Code Snippet A.3: Run default Selfish Mining attack

```
main.exe selfish --alpha 0.35 --blocks 1000 --runs 10
```

Code Snippet A.4: Customised Selfish Mining attack

Long-Range Attack (PoS):

```
main.exe longrange
```

Code Snippet A.5: Run default Long-Range attack

```
main.exe longrange --attack_frac 0.4 --initial_diff 5 --slots 1000 --runs 10
```

Code Snippet A.6: Customised Long-Range attack

Run Full Aggregate Analyses

```
main.exe aggregate selfish
main.exe aggregate sybil
main.exe aggregate longrange
```

Code Snippet A.7: Run full parameter sweeps with plots

Parameter Descriptions

`--consensus`	`pow` or `pos` (Sybil only)
`--attack_frac`	Attacker's resource fraction (e.g., 0.3 = 30%)
`--honest_nodes`	Number of honest nodes (Sybil only)
`--sybil_nodes`	Number of attacker-controlled fake identities
`--rounds`	Consensus rounds to simulate (Sybil only)
`--alpha`	Attacker's hash power share (Selfish only)
`--blocks`	Number of blocks to simulate (Selfish only)
`--initial_diff`	Honest chain's lead (Long-range only)
`--slots`	Time slots to simulate (Long-range only)
`--runs`	Number of Monte Carlo repetitions

Output

Simulation results are saved in the `results/` folder. Each run will create:

- A `.csv` file with raw results
- A `.png` plot visualising the results

These are generated automatically.