Part I: Literature Review

1. **Oleksandr Kuznetsov , Alex Rusnak, Anton Yezhov , Kateryna Kuznetsova, Dzianis Kanonik, Oleksandr Domin, Merkle Trees in Blockchain: A Study of Collision Probability and Security Implications, February 6, 2024.**

   Blockchain systems frequently use Merkle Trees to efficiently provide proof of data inclusion in datasets.They use cryptographic hashing to create a data structure in which the Merkle Root, the base of the tree, acts as an aggregate representation of the whole dataset, and each leaf is a hash of the leaves that came before it. With the use of this technique, it is possible to confirm whether a certain data element exists in a Merkle Tree without needing to visit the tree as a whole. Any participant in the system can independently calculate and validate the hash of the relevant data by supplying a path from the leaf to the root. Merkle trees are used extensively, yet little is known about their resilience against preimage attacks and collision resistance. Through a careful blending of theoretical research and empirical validation, the authors hope to close this gap. They carefully examine the likelihood of root collisions in Merkle trees, taking into account elements like hash length and path length in the tree. The research findings indicate a clear relationship between the lengthening of the path and the increased likelihood of root accidents. Longer pathways in the tree highlight possible weaknesses in security. On the other hand, a longer hash length dramatically lowers the chance of collisions, highlighting its vital role in bolstering security. Researchers and blockchain developers can benefit greatly from the insights this study has yielded. Through an awareness of the relationship among hash length, path length, and collision probabilities, practitioners can improve the operational efficiency and security of blockchain-based systems. In conclusion, this research not only points out the information gaps but also offers a framework for examining the security implications of Merkle trees. It clarifies current trends and highlights how crucial strong collision-resistant methods are for blockchain systems.

2. **Saif Al-Kuwari, James H. Davenport, Russell J. Bradford, Cryptographic Hash Functions: Recent Design Trends and Security Notions, September, 2011**
   This paper provides an overview of the field of cryptographic hash functions, including a range of techniques such as MD5, SHA-1, and SHA-2.Pre-image resistance, second pre-image resistance, and impact resistance are some of the basic characteristics that are covered.The authors examine how hash function designs have changed over time. They investigate new developments, innovations, and trends.The report illustrates the state-of-the-art methodologies by finding patterns and shifts in design decisions.The study explores the foundational ideas of hash function security. It breaks down security models and formal definitions.The trade-offs between computational efficiency and security properties are also examined.Although the study doesn't specifically address Merkle Trees, blockchain systems can greatly benefit from its insights.In Merkle Trees, hash functions are essential for maintaining data integrity and providing effective proofs of inclusion. When implementing Merkle-based structures, blockchain developers can make well-informed decisions by comprehending the security concepts covered in the paper.All

things considered, this article is a great resource for learning about hash function security, current advancements, and how they affect blockchain technology. By bridging the knowledge gap between theoretical underpinnings and real-world implementations, it helps Merkle Trees in blockchain ecosystems in an indirect way.

3. **Ilya Mironov, Hash functions: Theory, attacks, and applications, October, 2005**

SHA-1 and MD5, for example, are hash functions that were originally created for specialized cryptographic systems that have unique security needs. Developers and protocol designers began to use these hash functions extensively throughout time, viewing them as mysterious black boxes with magical abilities. Both MD5 and SHA-1 appeared to be resistant to cryptanalysis up until 2004. Nevertheless, there was a spike in assaults on these common hash methods beginning at that point. The paper defines hash function security features and presents notation. Concepts such as {0, 1}n (set of binary strings of length n) and {0, 1}* (set of all finite binary strings) are among those included in notation. The way hash functions are used in practice and their theoretical definitions diverge. Hash functions translate random texts into binary strings with a set length in real life. The study separates hash functions into three security tiers. A deterministic function that accepts two inputs, an arbitrary-length string and a key of length l, is defined as a hash function f with an n-bit output and a l-bit key.The hash function H applied to input x with key k is represented by the notation Hk(x). This paper examines common uses of hash functions in practical programming scenarios.It covers recent attacks on particular functions, as well as generic and iterative hash function attacks.The emphasis is on collision-finding attacks, which jeopardize hash function integrity.

Part II: Discussion Questions

1. **Explain why hash collisions are a mathematical inevitability.**

When two distinct inputs provide the same hash value, this is known as a hash collision. Let's examine the mathematical reasoning behind these collisions.

Finite Output Space: The hash value is the fixed-size output produced by hash functions from an input, which is often a message. Usually, this output is a series of bits. There are only a finite number of possible hash values since the output space is finite.

Finite Outputs, unlimited Inputs: A finite set of hash values is mapped to an unlimited set of possible inputs (messages) using hash functions. Consequently, a hash value may map to several different inputs.

Pigeonhole Principle: Think of the inputs as pigeons and the hash function as a series of pigeonholes, or potential hash values. There must be more than one pigeon in some pigeonholes if there are more pigeons (inputs) than pigeonholes (possible hash values). Similarly, collisions are inevitable if there are more unique inputs than there are potential hash values.

Probability and Randomness: As the number of inputs rises, the likelihood of collisions rises as well, even with carefully crafted hash algorithms. This is due to the fact that hash functions seek to evenly disperse inputs over the output space; yet, collisions may occur due to statistical patterns and unpredictability.

2. **Considering a room with N people, including Trudy, what's the probability that at least one other person shares Trudy's birthday? At what minimum N does this probability exceed 50%?**

The birthday problem is a captivating mathematical phenomena that pertains to the likelihood of two or more persons in a group of randomly selected individuals having the same birthday. Interestingly, the chance of this happening approaches 50% when there are just 23 persons.

Calculating Probability: Let's say that event A is the likelihood of finding a group of 23 individuals who don't have any birthdays in common.

The likelihood of finding a group of 23 people in which at least two of them have the same birthdate is represented by the complimentary event B.

P(B) = 1 - P(A) is what we have.

From the perspective of permutations: one can compute (P(A)) by taking into account the total number of birthdays that can occur without repetitions and where order is important (for example, mm/dd for a group of two people).

Comparably, (P(B)) deals with the total number of birthdays where repetitions occur and where there is a need for order (for example, mm/dd can occur for two people).

For this reason, (P(A)) and (P(B)) both include permutations.

With 23 people, there are more than half of a year's worth of pairs to take into account (23 \times 22 / 2 = 253).

As a result, even with just 23 individuals, there is a higher than 50% likelihood that at least two of them will share a birthdate, according to the birthday paradox1.

In conclusion, the birthday problem shows how unlikely events can have unexpected results. Thus, there's a 50–50 chance that two respondents to a poll conducted among a random sample of 23 will have the same birthday2.

3. **In a room of N people ($N \leq 365$), what's the probability of any two sharing a birthday, and what's the minimum N for this probability to be over 50%?**

Calculating Probabilities: Let's define event A as the likelihood of discovering a group of 23 individuals who don't have the same birthday twice.The likelihood of finding a group of 23 individuals in which at least two of them have the same birthday is represented by Event B.

P(B) = 1 - P(A) is what we have.

P(A) is the ratio of all birthdays (with and without repetitions and order matters taken into account) divided by all birthdays (with and without repetitions and order matters).

As a result, permutations are involved in (P(A)) and (P(B)).

Every conceivable pair of people has their birthdays compared, yielding a significant number of pairs (253) for 23 individuals.

This logical conclusion emphasizes how large the likelihood of shared birthdays is, even when there are just 23 people.

Approximate Minimum N: We can utilize the square root of the total number of possible birthdays (365) to get a 50% chance of common birthdays.

Thus, for the probability to be greater than 50%, about 23 persons are required.

Real-world uses of the birthday paradox include hash collision risk assessment and cryptographic attacks like the birthday attack. Thus, keep in mind the surprise possibility of shared birthdays the next time you're in a room with twenty-three people.