cryptography

Article

Efficient and Universal Merkle Tree Inclusion Proofs via OR Aggregation

Oleksandr Kuznetsov [1,2,3,*], Alex Rusnak [1], Anton Yezhov [1], Dzianis Kanonik [1], Kateryna Kuznetsova [1] and Oleksandr Domin [1]

[1] Proxima Labs, 1501 Larkin Street, Suite 300, San Francisco, CA 94109, USA; alex@proxima.one (A.R.); anton@proxima.one (A.Y.); denis@proxima.one (D.K.); kateryna@proxima.one (K.K.); dyomin@proxima.one (O.D.)
[2] Faculty of Engineering, eCampus University, Via Isimbardi 10, 22060 Novedrate, Italy
[3] Department of Information and Communication Systems Security, V. N. Karazin Kharkiv National University, 4 Svobody Sq., 61022 Kharkiv, Ukraine
[*] Correspondence: kuznetsov@proxima.one or oleksandr.kuznetsov@uniecampus.it or kuznetsov@karazin.ua

Abstract: Zero-knowledge proofs have emerged as a powerful tool for enhancing privacy and security in blockchain applications. However, the efficiency and scalability of proof systems remain a significant challenge, particularly in the context of Merkle tree inclusion proofs. Traditional proof aggregation techniques based on AND logic suffer from a high verification complexity and data communication overhead, limiting their practicality for large-scale applications. In this paper, we propose a novel proof aggregation approach based on OR logic, which enables the generation of compact and universally verifiable proofs for Merkle tree inclusion. By adapting and extending the concept of OR composition from Sigma protocols, we achieve a proof size that is independent of the number of leaves in the tree, and verification can be performed using any single valid leaf hash. This represents a significant improvement over AND aggregation, which requires the verifier to process all leaf hashes. We formally define the OR aggregation logic; describe the process of generating universal proofs; and provide a comparative analysis that demonstrates the advantages of our approach in terms of proof size, verification data, and universality. Furthermore, we discuss the potential of combining OR and AND aggregation logics to create complex acceptance functions, enabling the development of expressive and efficient proof systems for various blockchain applications. The proposed techniques have the potential to significantly enhance the scalability, efficiency, and flexibility of zero-knowledge proof systems, paving the way for more practical and adaptive solutions in large-scale blockchain ecosystems.

Keywords: zero-knowledge proofs; Merkle trees; proof aggregation; OR logic; universal proofs; blockchain; scalability; efficiency; flexibility; complex acceptance functions

check for updates

Citation: Kuznetsov, O.; Rusnak, A.; Yezhov, A.; Kanonik, D.; Kuznetsova, K.; Domin, O. Efficient and Universal Merkle Tree Inclusion Proofs via OR Aggregation. *Cryptography* **2024**, *8*, 28. https://doi.org/10.3390/cryptography8030028

Academic Editors: Cheng-Chi Lee, Tuan-Vinh Le, Chun-Ta Li, Dinh-Thuan Do and Agbotiname Lucky Imoize

Received: 23 May 2024
Revised: 2 July 2024
Accepted: 3 July 2024
Published: 5 July 2024

1. Introduction

Zero-knowledge proofs (ZKPs) have garnered significant attention in recent years due to their ability to enhance privacy and security in various applications, particularly in the domain of blockchain technology [1,2]. ZKPs allow one party (the prover) to convince another party (the verifier) that a statement is true without revealing any additional information beyond the validity of the statement itself [3]. This property makes ZKPs a powerful tool for enabling secure and privacy-preserving transactions, smart contracts, and other applications in blockchain systems [4–7].

One of the fundamental building blocks of many blockchain protocols is the Merkle tree [8–10], which is a data structure that enables the efficient and secure verification of large datasets. Merkle trees are used to store transactions, account balances, and other critical information in a compact and tamper-evident manner [8]. To prove the inclusion of a specific data element within a Merkle tree, a prover must provide a Merkle proof, which consists of a path of hashes from the leaf node (representing the data element) to the root of the tree [8–10].

However, the efficiency of Merkle proofs becomes a critical issue when dealing with large-scale blockchain systems. Specifically, we address the following problem:

- For a given set X of leaves in a Merkle tree, create a universal proof that allows for efficient verification of whether an arbitrary pair (b, h) belongs to X, where h is the hash value of b, without the need to provide or process all leaves from X during each verification.

This challenge is particularly relevant in scenarios where selective verification of individual leaves is required, such as in decentralized exchanges or supply chain management systems, where the ability to efficiently prove the inclusion of specific transactions or items without revealing the entire dataset is crucial.

While Merkle trees offer efficient verification for individual elements, proving the inclusion of multiple elements or generating universal proofs for all elements in the tree remains a challenge. This limitation becomes particularly apparent in scenarios that require frequent verifications or deal with large-scale datasets, where the cumulative overhead of multiple Merkle proofs can impact the system performance [11–13].

Traditional proof aggregation techniques based on AND logic, where multiple proofs are combined into a single proof, were proposed to address this issue [14,15]. However, these methods often result in increased verification complexity and data communication overhead, especially for large Merkle trees, as they require processing all leaves during verification. Recent work has explored alternative aggregation strategies, including the use of OR logic in the context of Sigma protocols [16,17]. Building upon these foundations, we propose a novel application of recursive OR aggregation specifically tailored for Merkle tree proofs, which allows for the efficient verification of individual leaves without the need to process the entire dataset.

In this paper, we present a practical approach to compressing Merkle proofs into a single, compact zero-knowledge proof using recursive OR aggregation. Our method enables the generation of a universal proof that can attest to the inclusion of any leaf in the Merkle tree, significantly reducing the overall proof size and verification complexity. This approach is particularly valuable in blockchain systems, where efficient proof generation and verification are crucial for scalability and performance.

The key contributions of our work are as follows:

1. We adapt and extend the concept of OR aggregation, which was previously discussed in the context of Sigma protocols, to create a recursive aggregation scheme specifically designed for Merkle tree proofs.
2. We provide a detailed description of the process for generating a universal, compact proof for Merkle tree inclusion using recursive OR aggregation.
3. We present a comparative analysis that demonstrates the advantages of our approach in terms of proof size, verification data, and universality, particularly in contrast to traditional AND aggregation methods.
4. We discuss the practical implications of our method for blockchain applications, including potential optimizations for smart contract execution and improvements in the overall system efficiency.

The rest of this paper is organized as follows: Section 2 provides the necessary background on zero-knowledge proofs, Merkle trees, and existing proof aggregation techniques. Section 3 introduces our proposed recursive OR aggregation scheme for Merkle tree proofs, including the formal definitions and the process of generating universal proofs. Section 4 presents a comparative analysis of our approach with traditional aggregation methods and discusses potential applications and extensions of our scheme. Finally, Section 5 concludes this paper and outlines future research directions.

2. Background and Literature Review

2.1. Zero-Knowledge Proofs

ZKPs are cryptographic protocols that allow a prover to convince a verifier that a statement is true without revealing any additional information beyond the validity of the statement [2]. The concept of ZKPs was first introduced by Goldwasser, Micali, and Rackoff in 1985 [3], and since then, it has been extensively studied and applied in various domains, including authentication, digital signatures, and blockchain technology [18,19].

A ZKP protocol must satisfy three main properties [20]:

1. Completeness: If the statement is true, an honest prover should be able to convince an honest verifier of its validity.
2. Soundness: If the statement is false, no prover (even a dishonest one) should be able to convince an honest verifier that it is true, except with a negligible probability.
3. Zero-knowledge: The verifier should not learn any information from the proof except for the validity of the statement.

In recent years, significant advancements have been made in the development of efficient ZKP systems, particularly in the context of blockchain applications. These include zk-SNARKs (zero-knowledge succinct non-interactive arguments of knowledge) [1,21] and zk-STARKs (zero-knowledge scalable transparent arguments of knowledge) [22–24]. These constructions have enabled privacy-preserving applications, such as confidential transactions [25], anonymous voting [26], and verifiable computation [27], in blockchain systems.

2.2. Merkle Trees

Merkle trees, also known as hash trees, are a fundamental data structure used in many blockchain protocols to enable the efficient and secure verification of large datasets [28]. A Merkle tree is a binary tree in which each leaf node contains the hash of a data block, and each non-leaf node contains the hash of its child nodes' hashes [28,29]. The root of the tree is a single hash value that represents the entire dataset.

The primary advantage of Merkle trees lies in their ability to provide efficient proofs of inclusion for individual elements without requiring the verifier to process the entire dataset [30]. This property is particularly valuable in blockchain systems, where it enables light clients to verify transactions without downloading the full blockchain [31].

To prove the inclusion of a data element in a Merkle tree, a prover needs to provide a Merkle proof, which consists of the hashes along the path from the leaf node (representing the data element) to the root of the tree. The verifier can then reconstruct the root hash using the provided hashes and compare it with the known root hash to verify the inclusion of the data element [30].

While Merkle trees offer efficient verification for individual elements, the cumulative cost of generating and verifying multiple proofs can become significant in scenarios involving large-scale data or frequent verifications. This issue has led researchers to explore various optimization techniques and alternative proof structures [11–13].

2.3. Proof Aggregation Techniques

As blockchain networks scale and the volume of data stored in Merkle trees grows, the efficiency of proof generation and verification has become an increasingly important consideration. To address this challenge, various proof aggregation techniques were proposed [32,33].

The most common proof aggregation approach is based on AND logic, where the aggregated proof is considered valid only if all the constituent proofs are valid [32,33]. In the context of Merkle tree inclusion proofs, AND aggregation allows the prover to combine the proofs for multiple data elements into a single proof. However, the verifier still needs to process all the leaf hashes to validate the aggregated proof, leading to high verification complexity, especially for large Merkle trees.

Recent research has explored alternative aggregation strategies to overcome the limitations of AND-based approaches. Notable among these is the concept of OR aggregation, which has been discussed in the context of Sigma protocols [16,17]. OR aggregation allows for the construction of proofs that are valid if at least one of the constituent proofs is valid, potentially offering advantages in terms of proof size and verification efficiency.

Other proof aggregation techniques explored in the literature include the following:

- Batch verification [6,23]: this approach allows for the simultaneous verification of multiple signatures or proofs, reducing the overall computational cost.
- Recursive proof composition [1,12]: this technique involves using the output of one proof as an input to another, enabling the construction of more complex proofs from simpler building blocks.
- Probabilistic proof aggregation [34,35]: these methods use probabilistic techniques to reduce the proof size and verification time, often at the cost of introducing a small probability of error.

A particularly relevant work in this context is the Maru project [36], which proposes an approach for embedding Merkle path elements into proofs. While this method offers improvements in terms of proof size and verification efficiency, it results in proofs that are specific to individual leaves rather than universal for the entire tree.

Our work builds upon these foundations, particularly the concept of OR aggregation, and extends it to create a recursive aggregation scheme specifically tailored for Merkle tree proofs. By doing so, we aimed to address the limitations of existing approaches and provide a more efficient and flexible solution for generating compact, universal proofs of inclusion in Merkle trees.

3. Enhanced Aggregation Logic

Before introducing our enhanced aggregation logic for Merkle tree proofs, it is crucial to establish the foundations upon which our work is built. We begin by reviewing key concepts from Sigma protocols, which form the basis for many zero-knowledge proof systems.

In the context of this paper, "aggregation" refers to the process of combining multiple individual proofs or data elements into a single coherent structure that can be verified as a whole. Specifically, in the realm of zero-knowledge proofs within Merkle trees, aggregation aims to consolidate numerous individual proofs of inclusion into a unified proof. This unified proof not only asserts the validity of multiple data elements concurrently but also optimizes the computational and communication overhead associated with their verification. We utilized OR aggregation logic, where a single composite proof is deemed valid if at least one of its constituent proofs holds true. This method contrasts with AND aggregation, which requires all constituent proofs to be valid for the composite proof to be accepted, and typically involves higher complexity and resource demands.

3.1. Foundations: Sigma Protocols and OR Composition

Sigma protocols, which were introduced by Cramer et al. [37], are three-move public coin protocols that allow a prover to convince a verifier of the validity of a statement without revealing any additional information. A Sigma protocol Π for a relation R consists of algorithms (P_1, P_2, V), where the following occurs:

1. $P_1(x, w) \rightarrow a$: the prover's first move, which generates the initial message a.
2. $V(x, a) \rightarrow c$: the verifier's challenge c.
3. $P_2(x, w, a, c) \rightarrow z$: the prover's response to the challenge.
4. $V(x, a, c, z) \rightarrow \{0, 1\}$: the verifier's final decision to accept (1) or reject (0).

Sigma protocols possess three key properties:

1. Completeness: an honest prover can always convince an honest verifier.
2. Special soundness: given two accepting transcripts (a, c, z) and (a, c', z') for $c \neq c'$, one can efficiently extract a witness w.
3. Special honest-verifier zero knowledge: there exists a simulator that can produce transcripts indistinguishable from real protocol executions.

Building upon Sigma protocols, Cramer et al. [37] introduced the OR composition technique, which allows for proving knowledge of at least one witness among multiple statements. This technique forms the theoretical basis for our approach to Merkle tree proof aggregation.

Now, we introduce our novel approach to proof aggregation in zero-knowledge proof systems for Merkle trees, which addresses the limitations of traditional AND aggregation logic. Our enhanced aggregation scheme, which is based on OR logic, enables the generation of compact and universally verifiable zk-proofs for Merkle tree inclusion.

3.2. Motivation for an Improved Universal Proof

Let $\mathcal{M}$ be a Merkle tree with n leaves, where $n = 2^d$ for some integer $d \geq 0$. Each leaf is associated with a data block b_i $(i = 1, \ldots, n)$ and the corresponding leaf hash is computed as $h_i = H(b_i)$, where $H(\cdot)$ is a cryptographic hash function. The Merkle tree is constructed by recursively hashing pairs of adjacent nodes until a single root hash h_{root} is obtained.

In traditional approaches, proving the inclusion of a leaf in a Merkle tree requires providing a path of hashes from the leaf to the root. While this is efficient for single-leaf verification, this method becomes cumbersome when proving the inclusion of multiple leaves or when generating a universal proof for all leaves.

To address this, previous work explored proof aggregation techniques. The most common approach is based on AND logic, where an aggregated zk-proof π_{AND} is considered valid only if all constituent zk-proofs $\pi_1, \ldots, \pi_m$ are valid. Formally (Figure 1),

$$\pi_{\text{AND}} = \text{AND}(\pi_1, \ldots, \pi_m) \Leftrightarrow \mathcal{V}(\pi_{\text{AND}}, (h_1, \ldots, h_m)) \equiv \wedge_{j=1}^{m} \mathcal{V}(\pi_j, h_j).$$

where $\mathcal{V}(\pi_i, h_i)$ denotes the verification function that outputs 1 if π_i is a valid proof for h_i and 0 otherwise.

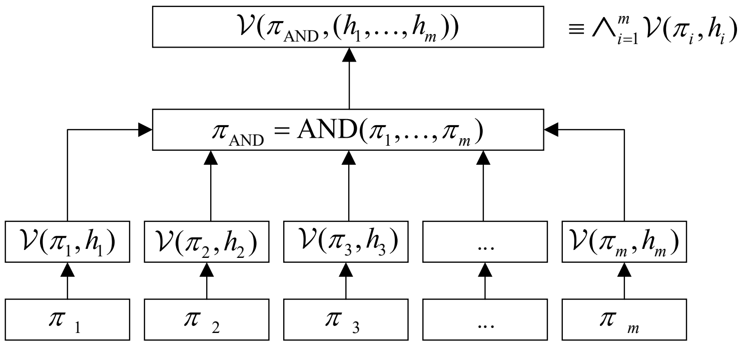

$\mathcal{V}(\pi_{\text{AND}}, (h_1, \ldots, h_m))$
$\equiv \wedge_{i=1}^{m} \mathcal{V}(\pi_i, h_i)$
$\pi_{\text{AND}} = \text{AND}(\pi_1, \ldots, \pi_m)$
$\mathcal{V}(\pi_1, h_1)$
$\mathcal{V}(\pi_2, h_2)$
$\mathcal{V}(\pi_3, h_3)$
...
$\mathcal{V}(\pi_m, h_m)$
π_1
π_2
π_3
...
π_m

Figure 1. Aggregation logic "AND" of zero-knowledge proofs.

While AND aggregation has been effective in various scenarios, it poses significant challenges when applied to large Merkle trees. The main issue is verification complexity: the verifier needs to process all leaf hashes to validate the proof, leading to high computational and communication overhead for large trees.

To illustrate this, consider the problem of proving the inclusion of a single leaf b_i in a Merkle tree $\mathcal{M}$. In a standard Merkle proof, the prover provides the verifier with a path of hashes from the leaf b_i to the root h_{root}, along with the corresponding sibling hashes at each level. The verifier can then recompute the root hash and compare it with the known value to verify the inclusion of b_i.

However, if we were to use AND aggregation to create a single zk-proof for the inclusion of l_i (highlighted in yellow in Figure 2), the prover would need to provide proofs for all the leaves in the tree, i.e., $\pi_1, \ldots, \pi_n$, where $n = 2^d$. The aggregated proof π_{AND} would then be validated by verifying each constituent proof (Figure 2):

$$\mathcal{V}(\pi_{\text{AND}}, (h_1, \ldots, h_n)) \equiv \wedge_{j=1}^{n} \mathcal{V}(\pi_j, h_j).$$

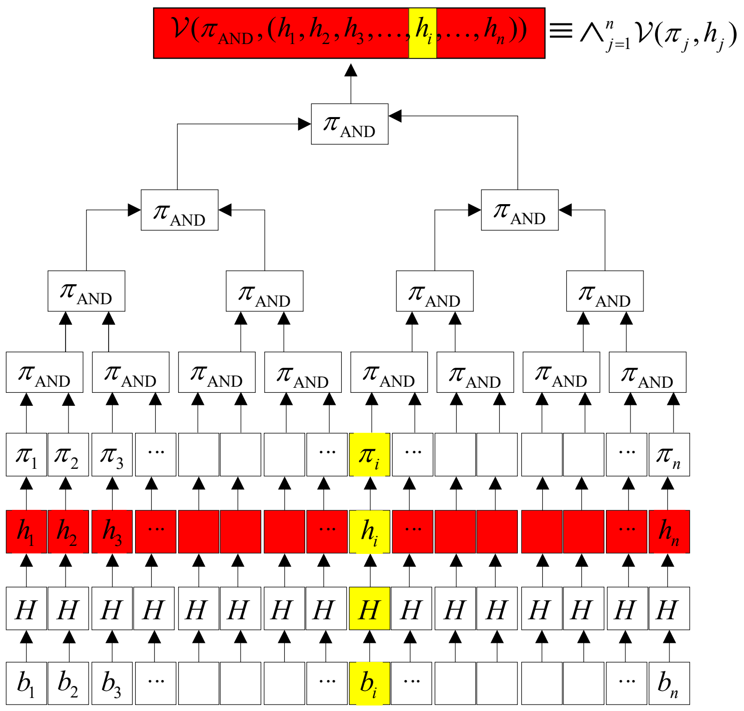

$\mathcal{V}(\pi_{\text{AND}},(h_1,h_2,h_3,\ldots,h_i,\ldots,h_n)) \equiv \bigwedge_{j=1}^{n}\mathcal{V}(\pi_j,h_j)$
π_{AND}
π_1 π_2 π_3 … … π_i … … π_n
h_1 h_2 h_3 … … h_i … … h_n
H
b_1 b_2 b_3 … … b_i … … b_n

Figure 2. AND logic to create a single zk-proof of inclusion.

The main challenge with using AND aggregation for Merkle tree inclusion proofs is the verification complexity. While the size of the aggregated proof π_{AND} itself may be compact, the verifier would need to be provided with all the leaf hashes $h_1, \ldots, h_n$ to validate the proof (highlighted in red in Figure 2). In a tree with 2^{30} leaves (corresponding to a 1 GB data block), this would require the prover to send and the verifier to process 2^{30} hash values, each of which is typically 256 bits long, resulting in a total communication overhead of 32 GB. This makes the verification process impractical for large Merkle trees.

One way to mitigate this issue is to embed the specific Merkle path elements for a particular leaf into the final proof, as was done in the Maru project [36]. This approach eliminates the need to provide all the leaf hashes during verification. However, the resulting proof is no longer universal, as it is tailored to prove the inclusion of a single, specific leaf. If the prover wants to demonstrate the inclusion of a different leaf, a new proof must be generated, embedding the corresponding Merkle path elements.

Formally, let $\pi_{\text{AND}}(b_i)$ denote the AND-aggregated proof for the inclusion of leaf b_i, with the Merkle path elements $h_1, h_2, \ldots, h_d$ for b_i embedded in the proof (highlighted in yellow in Figure 3). The verification of $\pi_{\text{AND}}(b_i)$ would only require the leaf hash h_i and the root hash h_{root} (highlighted in orange in Figure 3):

$$\mathcal{V}(\pi_{\text{AND}}(b_i), (h_i, h_{\text{root}})) = 1 \Leftrightarrow b_i \in \mathcal{M}.$$

Figure 3 shows the following:

- x—public statement (highlighted in green in Figure 3);
- w—secret witness (highlighted in red in Figure 3);
- $||$—concatenation function (combining vectors).

While this approach reduces the communication overhead and verification complexity compared with AND aggregation, it comes at the cost of proof universality. If the prover wants to demonstrate the inclusion of a different leaf, a new proof must be generated, embedding the corresponding Merkle path elements. Consequently, the prover must generate a separate proof $\pi_{\text{AND}}(b_i)$ for each leaf b_i ($i = 1, 2, \ldots, n$) they want to prove inclusion for. This can be inefficient in scenarios requiring frequent proof generation for different subsets of leaves or when dealing with a large number n of leaves in dynamic environments.

In contrast, our OR aggregation method addresses this limitation by creating a single, universal proof that can verify the inclusion of any leaf without requiring regeneration for different leaves or subsets. This approach maintains the efficiency of verification while providing greater flexibility and reducing the computational overhead for the prover in dynamic scenarios.

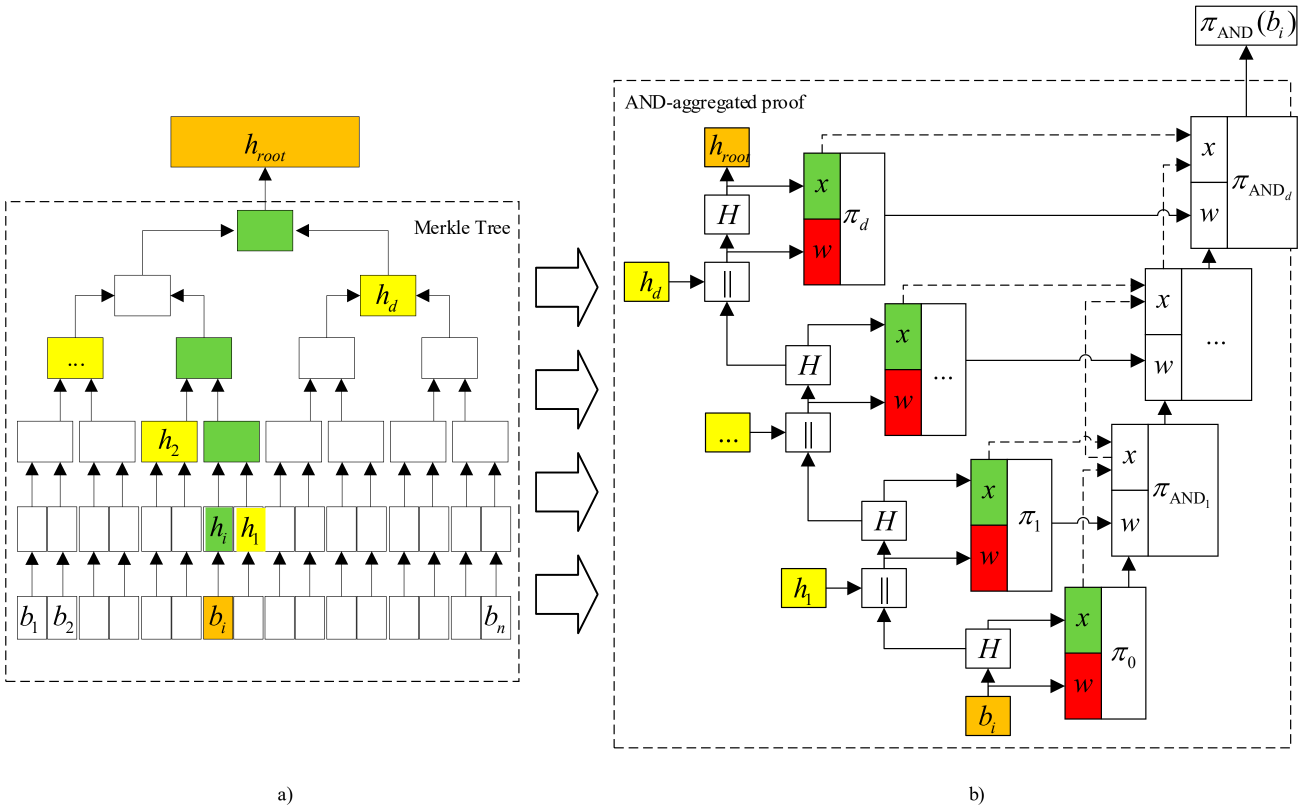


Figure 3. Logic for generating a single inclusion proof with Merkle path embedding (as used in the Maru project [36]): (**a**) Merkle tree; (**b**) Proof generation scheme.

3.3. OR Aggregation for Merkle Tree Proofs

Building upon the concept of OR composition in Sigma protocols, we propose an enhanced aggregation scheme based on OR logic specifically tailored for Merkle tree proofs. Our approach allows for the construction of a valid proof if at least one of the constituent proofs is valid, significantly reducing the verification complexity.

Formally, let $\pi_1, \ldots, \pi_m$ be proofs for the validity of leaf hashes $h_1, \ldots, h_m$, respectively. The OR aggregation of these proofs, denoted by π_{OR}, is defined as follows (Figure 4):

$$\pi_{\text{OR}} = \text{OR}(\pi_1, \ldots, \pi_m) \Leftrightarrow \mathcal{V}(\pi_{\text{OR}}, h_i) \equiv \vee_{j=1}^{m} \mathcal{V}(\pi_j, h_j).$$

This definition ensures that the aggregated proof π_{OR} is valid if and only if at least one of the constituent proofs $\pi_1, \ldots, \pi_m$ is valid. This property is crucial for our approach, as it allows for efficient verification using any single leaf. Specifically, if we supply any valid leaf hash $h_i \in h_1, \ldots, h_m$ to the proof-checking function $\mathcal{V}(\pi_{\text{OR}}, h_i)$, we obtain confirmation of inclusion for that leaf:

$$\mathcal{V}(\pi_{\text{OR}}, h_i) \equiv \vee_{j=1}^{m} \mathcal{V}(\pi_j, h_j) = 1, \text{ for } h_i \in \{h_1, \ldots, h_m\}.$$

This formulation demonstrates that our OR-aggregated proof can verify the inclusion of any leaf in the Merkle tree using a single, compact proof.

The OR aggregation logic enables a more efficient traversal of the Merkle tree, where proofs for individual leaves can be aggregated in a way that naturally follows the tree structure.

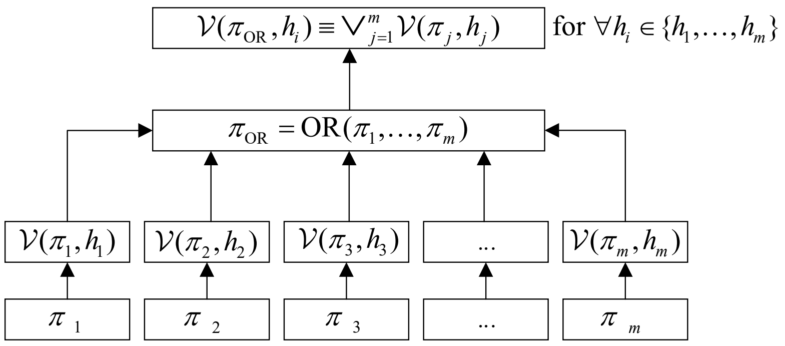
While our OR aggregation process follows a structure similar to the standard Merkle tree construction, it operates on proofs rather than hash values. This key distinction allows us to create a universal proof for leaf inclusion without modifying the underlying Merkle tree structure.

Let $\mathcal{M}$ be a Merkle tree with n leaves, and let $b_1, \ldots, b_n$ be the leaf nodes with corresponding hashes $h_1, \ldots, h_n$. The aggregation process begins at the leaf level and progresses upward, combining proofs for adjacent nodes to form aggregated proofs for their parent nodes. At each level, we apply our OR logic to the proofs:

$$\pi_{\mathrm{OR}_{Parent}} = \mathrm{OR}(\pi_{\mathrm{left}}, \pi_{\mathrm{right}}),$$

Here, $\pi_{\mathrm{OR}_{Parent}}$ is the aggregated proof for a parent node, which is derived from the proofs π_{left} and π_{right} of its left and right child nodes, respectively. This operation preserves the critical property that the aggregated proof remains valid if either of its constituent proofs is valid.

This approach directly addresses the challenge of efficient selective verification, allowing us to prove the inclusion of any leaf b_i with hash h_i in the Merkle tree using a single, compact proof. Unlike standard Merkle proofs, our method does not require providing the entire path from leaf to root for each verification.



$\mathcal{V}(\pi_{\mathrm{OR}}, h_i) \equiv \bigvee_{j=1}^{m} \mathcal{V}(\pi_j, h_j)$
for $\forall h_i \in \{h_1, \ldots, h_m\}$
$\pi_{\mathrm{OR}} = \mathrm{OR}(\pi_1, \ldots, \pi_m)$
$\mathcal{V}(\pi_1, h_1)$
$\mathcal{V}(\pi_2, h_2)$
$\mathcal{V}(\pi_3, h_3)$
...
$\mathcal{V}(\pi_m, h_m)$
π_1
π_2
π_3
...
π_m

Figure 4. Aggregation logic "OR" of zero-knowledge proofs.

3.4. Generating a Universal Proof for Merkle Tree Inclusion

Our OR aggregation scheme enables the generation of a universal proof that succinctly attests to the inclusion of any valid leaf in the Merkle tree. This process consists of the following steps (Figure 5):

1. Generate proofs for each leaf: For each leaf node b_i $(i = 1, \ldots, n)$ in the Merkle tree, generate a zero-knowledge proof π_i that attests to the correctness of the leaf hash h_i. This can be done using a suitable zero-knowledge proof system, such as zk-SNARKs or zk-STARKs.
2. Aggregate proofs using OR logic: Starting from the leaves, recursively aggregate the proofs of adjacent nodes using OR logic, as described in Section 4.2. At each level, the proofs of sibling nodes are combined to form a proof for their parent node (highlighted in yellow in Figure 5). This process is repeated until a single proof $\pi_{\mathrm{OR}_{\mathrm{root}}}$ is obtained for the root of the tree.
3. Output the universal proof: The aggregated proof for the root of the Merkle tree, $\pi_{\mathrm{OR}_{\mathrm{root}}}$, serves as the universal proof of inclusion. This proof has the property that it can be validated by providing any one of the valid leaf hashes as the input:

$$\mathcal{V}(\pi_{\mathrm{OR}_{\mathrm{root}}}, h_i) = 1 \Leftrightarrow l_i \in \mathcal{M}, \text{ for } i = 1, \ldots, n.$$

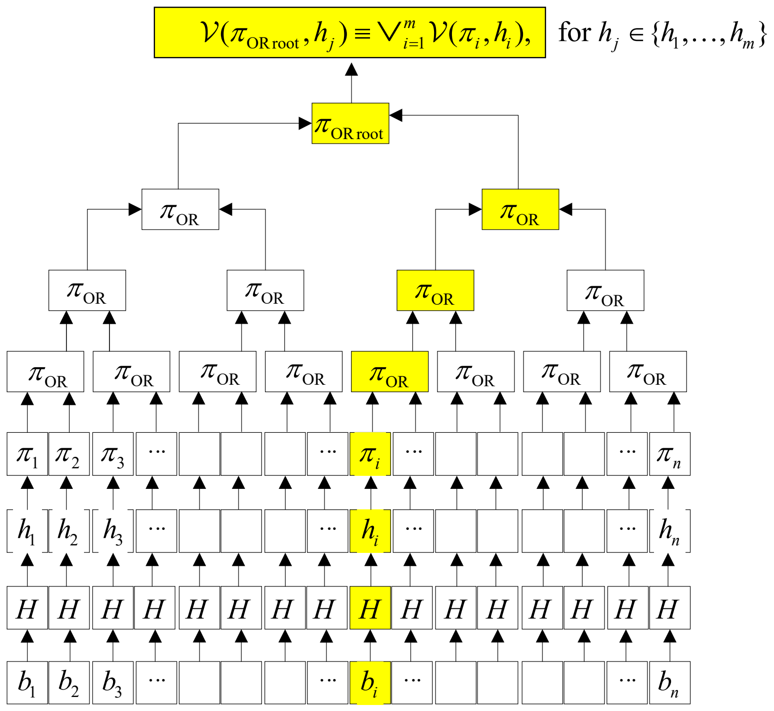

$\mathcal{V}(\pi_{\text{OR root}}, h_j) \equiv \bigvee_{i=1}^{m} \mathcal{V}(\pi_i, h_i)$, for $h_j \in \{h_1, \ldots, h_m\}$
$\pi_{\text{OR root}}$
π_{OR}
π_1 π_2 π_3 $\cdots$ π_i π_n
h_1 h_2 h_3 h_i h_n
H
b_1 b_2 b_3 b_i b_n

Figure 5. OR logic to create a single zk-proof of inclusion.

The resulting universal proof $\pi_{\text{OR}_{\text{root}}}$ is compact, as its size is independent of the number of leaves in the tree. Moreover, the proof can be efficiently verified by providing any one of the valid leaf hashes without requiring the prover to send all the leaf hashes or embed specific Merkle path elements for each leaf.

3.5. Comparison with Existing Approaches

Our OR aggregation scheme for Merkle tree proofs builds upon the theoretical foundations of Sigma protocols and OR composition techniques, while addressing the specific challenges of Merkle tree verification in blockchain systems. Unlike the approach used in the Maru project [36], which embeds Merkle path elements for a particular leaf, our method generates a truly universal proof that can be verified using any leaf in the tree.

Furthermore, our approach differs from traditional Sigma-protocol-based systems in its specific application to Merkle trees and its recursive nature. While standard OR composition allows for proving knowledge of one out of many witnesses, our scheme enables the aggregation of proofs across all levels of the Merkle tree, resulting in a single, compact proof for the entire structure.

By leveraging the efficiency of OR logic in this context, we achieve a significant reduction in proof size and verification complexity compared with AND-based aggregation methods, especially for large Merkle trees. This makes our approach particularly suitable for blockchain applications where efficient proof generation and verification are crucial for scalability and performance.

4. Discussion and Analysis

4.1. Comparative Analysis of Merkle Tree Proof Techniques

Our proposed OR aggregation logic for Merkle tree proof aggregation offers several advantages over traditional approaches. To quantify these benefits, we conducted a comprehensive comparative analysis of our method against standard Merkle proofs, AND aggregation, and the Maru project's approach [36].

Table 1 presents a summary of our findings, comparing key metrics across the four approaches.

Table 1. Comparative analysis of Merkle tree proof techniques, where n is the number of leaves in the Merkle tree.

Metric	**Standard Merkle Proof**	**AND Aggregation**	**Maru Project [36]**	**Our OR Aggregation**
Proof size	O(log n)	O(1)	O(1)	O(1)
Verification data	O(log n)	O(n)	O(1)	O(1)
Universality	No	Yes	No	Yes
Verification complexity	O(log n)	O(1)	O(1)	O(1)

The key observations from Table 1 are as follows:

1. Standard Merkle proof: while efficient for single-leaf verification, it lacks universality and scales logarithmically with tree size.
2. AND aggregation: offers a universal proof but requires all leaf hashes for verification, leading to a high data overhead.
3. Maru project [36]: achieves constant-size proofs and verification data but lacks universality.
4. Our OR aggregation: combines the advantages of constant-size proofs, minimal verification data, and universality.

4.2. Practical Implications for Blockchain Systems

The efficiency gains provided by our OR aggregation technique have several practical implications for blockchain systems:

1. Improved throughput: By reducing the verification complexity to O(1), our approach allows for significantly higher transaction throughput in blockchain networks. This is particularly important for large-scale, high-volume applications.
2. Reduced storage requirements: the compact nature of our universal proofs means that less storage is required for maintaining proof data, potentially leading to reduced costs for node operators.
3. Enhanced light client functionality: our method enables more efficient light client implementations, as clients can verify the inclusion of any leaf in the Merkle tree with minimal computational and data transfer overhead.
4. Flexible verification: the ability to verify the inclusion of any leaf using a single universal proof provides greater flexibility in how blockchain data can be accessed and verified.

4.3. Extending the Technique to New Applications

The introduction of OR aggregation logic alongside traditional AND aggregation opens up new possibilities for constructing complex acceptance functions at the proof-generation level. By combining these aggregation functions, we can create sophisticated proof systems that cater to various business logic requirements in blockchain applications.

For instance:

1. Partial group verification: in scenarios where a condition must be met by at least one participant from a group, OR aggregation can be used to efficiently verify this without checking each proof individually.
2. Complete group verification: for cases requiring all participants to satisfy a condition, AND aggregation can be employed to create a single, verifiable proof of complete compliance.
3. Nested conditions: complex scenarios involving combinations of conditions (e.g., “all participants from group A OR at least one from group B”) can be represented by nesting AND and OR aggregations.

This flexibility in constructing acceptance functions at the proof level can significantly enhance the expressiveness and efficiency of blockchain applications. It allows for the offloading of complex verification logic from smart contracts to the proof generation phase, potentially leading to more streamlined and cost-effective contract execution.

4.4. Potential Limitations and Future Work

While our OR aggregation technique offers significant advantages, it is important to acknowledge potential limitations and areas for future research:

1. Proof generation overhead: Although verification is highly efficient, the initial proof generation process may be more computationally intensive than traditional methods. Future work could focus on optimizing this process.
2. Security considerations: As with any new cryptographic technique, thorough security analysis is crucial. Future studies should focus on formal security proofs and potential attack vectors.
3. Integration with existing systems: further research is needed to explore the best practices for integrating our approach with existing blockchain protocols and infrastructure.
4. Extension to other data structures: while our focus has been on Merkle trees, future work could explore the application of similar OR aggregation techniques to other cryptographic data structures used in blockchain systems.
5. Theoretical foundations: further research could explore the theoretical underpinnings of our approach, potentially leading to new insights in the field of zero-knowledge proofs and their applications.

In conclusion, our OR aggregation technique for Merkle tree proofs represents a significant advancement in the field of blockchain scalability and efficiency. By enabling constant-time verification and compact universal proofs, our approach addresses key limitations of existing methods and opens new possibilities for high-performance blockchain applications. As the blockchain ecosystem continues to evolve, techniques like ours will play a crucial role in enabling the next generation of scalable, efficient, and secure distributed systems.

5. Conclusions

In this paper, we introduce a novel proof-aggregation technique based on OR logic, which addresses the limitations of traditional AND aggregation in the context of Merkle tree inclusion proofs. Our approach, which builds upon and extends the OR composition concept from Sigma protocols, enables the generation of compact and universally verifiable proofs, allowing for efficient and scalable verification of Merkle tree inclusion.

We formally defined the OR aggregation logic and described the process of generating a universal proof for Merkle tree inclusion using this approach. The resulting proof is not only compact in size but also universal, capable of being verified using any single valid leaf hash. This provides a significant advantage over traditional Merkle proofs and AND aggregation methods, particularly for large-scale blockchain applications.

Through a comparative analysis, we demonstrated the benefits of our proposed approach in terms of the proof size, verification data, and universality. Our OR aggregation scheme achieves constant-size proofs and verification data, regardless of the size of the Merkle tree. This represents a substantial improvement over standard Merkle proofs, which scale logarithmically, and AND aggregation, which requires linear growth in verification data.

Furthermore, we discuss the potential of combining OR and AND aggregation logics to create complex acceptance functions at the proof generation level. This flexibility enables the development of expressive and efficient proof systems that can cater to various business logic requirements in blockchain applications. While our approach offers substantial benefits, we acknowledge that there are areas for future research and potential limitations to address. These include optimizing the proof-generation process, conducting thorough security analyses, and exploring integration strategies with existing blockchain protocols.

The proposed techniques have the potential to significantly enhance the scalability, efficiency, and expressiveness of zero-knowledge proof systems in the context of Merkle tree inclusion proofs and beyond. As the adoption of zero-knowledge proofs continues to grow in blockchain applications, the ability to construct flexible and efficient proof aggregation schemes will be crucial in enabling the development of scalable and practical solutions.

In conclusion, our OR aggregation technique for Merkle tree proofs represents a significant step forward in addressing the scalability and efficiency challenges faced by current blockchain systems. By enabling constant-time verification and compact universal proofs, our approach opens new possibilities for high-performance blockchain applications and contributes to the ongoing evolution of secure and scalable distributed systems.

Author Contributions: Conceptualization, methodology, and writing—original draft preparation, O.K.; funding acquisition and resources, A.R.; project administration, A.Y.; data curation and validation, D.K.; software, validation, and writing—review and editing, K.K.; investigation and formal analysis, O.D. All authors have read and agreed to the published version of the manuscript.

Funding: This project received funding from the Proxima Labs, 1501 Larkin Street, suite 300, San Francisco, USA.

Data Availability Statement: The original contributions made by this study are included in this article, further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest. The authors were employed by the company Proxima Labs, 1501 Larkin Street, suite 300, San Francisco, USA. The authors declare that this study received funding from Proxima Labs. The funder was not involved in the study design; collection, analysis, or interpretation of the data; the writing of this article; or the decision to submit it for publication.

References

1. Bowe, S.; Gabizon, A.; Green, M.D. A Multi-Party Protocol for Constructing the Public Parameters of the Pinocchio Zk-SNARK. In Proceedings of the Financial Cryptography and Data Security: FC 2018 International Workshops, BITCOIN, VOTING, and WTSC, Nieuwpoort, Curaçao, 26 February–2 March 2018.
2. Ben-Sasson, E.; Chiesa, A.; Tromer, E.; Virza, M. Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture. In Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, 20–22 August 2014.
3. Goldwasser, S.; Micali, S.; Rackoff, C. The Knowledge Complexity of Interactive Proof-Systems. In Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing, Providence, RI, USA, 6–8 May 1985; pp. 291–304.
4. Zhou, L.; Diro, A.; Saini, A.; Kaisar, S.; Hiep, P.C. Leveraging Zero Knowledge Proofs for Blockchain-Based Identity Sharing: A Survey of Advancements, Challenges and Opportunities. *J. Inf. Secur. Appl.* **2024**, *80*, 103678. [CrossRef]
5. Shahrouz, J.K.; Analoui, M. An Anonymous Authentication Scheme with Conditional Privacy-Preserving for Vehicular Ad Hoc Networks Based on Zero-Knowledge Proof and Blockchain. *Ad Hoc Netw.* **2024**, *154*, 103349. [CrossRef]
6. Ren, Z.; Yan, E.; Chen, T.; Yu, Y. Blockchain-Based CP-ABE Data Sharing and Privacy-Preserving Scheme Using Distributed KMS and Zero-Knowledge Proof. *J. King Saud Univ.-Comput. Inf. Sci.* **2024**, *36*, 101969. [CrossRef]
7. Zheng, H.; You, L.; Hu, G. A Novel Insurance Claim Blockchain Scheme Based on Zero-Knowledge Proof Technology. *Comput. Commun.* **2022**, *195*, 207–216. [CrossRef]
8. Ethereum Ethereum Yellow Paper. 2023. Available online: https://github.com/ethereum/yellowpaper (accessed on 2 July 2024).
9. Mitra, D.; Tauz, L.; Dolecek, L. Graph Coded Merkle Tree: Mitigating Data Availability Attacks in Blockchain Systems Using Informed Design of Polar Factor Graphs. *IEEE J. Sel. Areas Inf. Theory* **2023**, *4*, 434–452. [CrossRef]
10. Mitra, D.; Tauz, L.; Dolecek, L. Polar Coded Merkle Tree: Improved Detection of Data Availability Attacks in Blockchain Systems. In Proceedings of the 2022 IEEE International Symposium on Information Theory (ISIT), Espoo, Finland, 26 June–1 July 2022; pp. 2583–2588.
11. Otte, P.; de Vos, M.; Pouwelse, J. TrustChain: A Sybil-Resistant Scalable Blockchain. *Future Gener. Comput. Syst.* **2020**, *107*, 770–780. [CrossRef]
12. Nasir, M.H.; Arshad, J.; Khan, M.M.; Fatima, M.; Salah, K.; Jayaraman, R. Scalable Blockchains—A Systematic Review. *Future Gener. Comput. Syst.* **2022**, *126*, 136–162. [CrossRef]
13. Bin Hasan, K.M.; Sajid, M.; Lapina, M.A.; Shahid, M.; Kotecha, K. Blockchain Technology Meets 6 G Wireless Networks: A Systematic Survey. *Alex. Eng. J.* **2024**, *92*, 199–220. [CrossRef]

14. Singh, R.; Dwivedi, A.D.; Mukkamala, R.R.; Alnumay, W.S. Privacy-Preserving Ledger for Blockchain and Internet of Things-Enabled Cyber-Physical Systems. *Comput. Electr. Eng.* **2022**, *103*, 108290. [CrossRef]
15. Ràfols, C.; Zacharakis, A. Folding Schemes with Selective Verification. In Proceedings of the 8th International Conference on Cryptology and Information Security in Latin America, Quito, Ecuador, 3–6 October 2023.
16. Ciampi, M.; Persiano, G.; Scafuro, A.; Siniscalchi, L.; Visconti, I. Improved OR-Composition of Sigma-Protocols. In Proceedings of the Theory of Cryptography: 13th International Conference, TCC 2016-A, Tel Aviv, Israel, 10–13 January 2016; Kushilevitz, E., Malkin, T., Eds.; Springer: Berlin/Heidelberg, Germany, 2016; pp. 112–141.
17. Nitulescu, A. A Gentle Introduction to SNARKs. 2019. Available online: https://api.semanticscholar.org/CorpusID:209520686 (accessed on 2 July 2024).
18. Ramos Fernández, R. Evaluation of Trust Service and Software Product Regimes for Zero-Knowledge Proof Development under eIDAS 2.0. *Comput. Law Secur. Rev.* **2024**, *53*, 105968. [CrossRef]
19. Wen, X.-J.; Chen, Y.-Z.; Fan, X.-C.; Zhang, W.; Yi, Z.-Z.; Fang, J.-B. Blockchain Consensus Mechanism Based on Quantum Zero-Knowledge Proof. *Opt. Laser Technol.* **2022**, *147*, 107693. [CrossRef]
20. Boneh, D.; ZK Whiteboard Sessions. Introductory Modules with Dan Boneh. ZK Hack. Available online: https://zkhack.dev/whiteboard/ (accessed on 2 July 2024).
21. Bowe, S.; Gabizon, A.; Miers, I. Scalable Multi-Party Computation for Zk-SNARK Parameters in the Random Beacon Model. *Cryptol. ePrint Arch.* 2017; *preprint*.
22. Ashur, T.; Dhooghe, S. MARVELlous: A STARK-Friendly Family of Cryptographic Primitives. *Cryptol. ePrint Arch.* 2018; *preprint*.
23. Boneh, D.; Bünz, B.; Fisch, B. Batching Techniques for Accumulators with Applications to IOPs and Stateless Blockchains. In Proceedings of the Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, 18–22 August 2019; Boldyreva, A., Micciancio, D., Eds.; Springer International Publishing: Cham, Swizterland, 2019; pp. 561–586.
24. Garreta, A.; Hovhanissyan, H.; Jivanyan, A.; Manzur, I.; Villalobos, I.; Zając, M. On Amortization Techniques for FRI-Based SNARKs. *IACR Cryptol. ePrint Arch.* 2024; *preprint*.
25. Huang, Y.; Zheng, X.; Zhu, Y. Optimized CPU–GPU Collaborative Acceleration of Zero-Knowledge Proof for Confidential Transactions. *J. Syst. Archit.* **2023**, *135*, 102807. [CrossRef]
26. Emami, A.; Yajam, H.; Akhaee, M.A.; Asghari, R. A Scalable Decentralized Privacy-Preserving e-Voting System Based on Zero-Knowledge off-Chain Computations. *J. Inf. Secur. Appl.* **2023**, *79*, 103645. [CrossRef]
27. Yao, S.; Zhang, D. An Anonymous Verifiable Random Function with Unbiasability and Constant Size Proof. *J. Inf. Secur. Appl.* **2024**, *83*, 103778. [CrossRef]
28. Merkle, R.C. Method of Providing Digital Signatures. 1982. Available online: https://patents.google.com/patent/US4309569A/en (accessed on 2 July 2024).
29. Merkle, R.C. A Digital Signature Based on a Conventional Encryption Function. In Proceedings of the Advances in Cryptology—CRYPTO'87, Conference on the Theory and Application of Cryptographic Techniques, Santa Barbara, CA, USA, 16–20 August 1987; Pomerance, C., Ed.; Springer: Berlin, Heidelberg, 1988; pp. 369–378.
30. Kuznetsov, O.; Rusnak, A.; Yezhov, A.; Kuznetsova, K.; Kanonik, D.; Domin, O. Merkle Trees in Blockchain: A Study of Collision Probability and Security Implications. *Internet Things* **2024**, *26*, 101193. [CrossRef]
31. George, J.T. Bitcoin. In *Introducing Blockchain Applications: Understand and Develop Blockchain Applications Through Distributed Systems*; George, J.T., Ed.; Apress: Berkeley, CA, USA, 2022; pp. 9–54, ISBN 978-1-4842-7480-4.
32. ZKP MOOC Lecture 10: Recursive SNARKs. 2023. Available online: https://www.youtube.com/watch?v=0LW-qeVe6QI (accessed on 2 July 2024).
33. StarkWare Recursive STARKs. StarkWare. 2022. Available online: https://medium.com/starkware/recursive-starks-78f8dd401025 (accessed on 2 July 2024).
34. Bellare, M.; Garay, J.A.; Rabin, T. Fast Batch Verification for Modular Exponentiation and Digital Signatures. In Proceedings of the Advances in Cryptology—EUROCRYPT'98, International Conference on the Theory and Applications of Cryptographic Techniques, Espoo, Finland, 31 May–4 June 1998; Nyberg, K., Ed.; Springer: Berlin/Heidelberg, Germany, 1998; pp. 236–250.
35. Waters, B.; Wu, D.J. Batch Arguments for NP and More from Standard Bilinear Group Assumptions. In Proceedings of the Advances in Cryptology—CRYPTO 2022, Annual International Cryptology Conference, Santa Barbara, CA, USA, 15–18 August 2022.
36. Kuznetsov, O.; Rusnak, A.; Yezhov, A.; Kanonik, D.; Kuznetsova, K.; Karashchuk, S. Enhanced Security and Efficiency in Blockchain with Aggregated Zero-Knowledge Proof Mechanisms. *IEEE Access* **2024**, *12*, 49228–49248. [CrossRef]
37. Cramer, R.; Damgård, I.; Schoenmakers, B. Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In Proceedings of the Advances in Cryptology—CRYPTO'94, Annual International Cryptology Conference, Santa Barbara, CA, USA, 21–25 August 1994; Desmedt, Y.G., Ed.; Springer: Berlin/Heidelberg, Germany, 1994; pp. 174–187.