Enhancing Credential Revocation Using zk-SNARK and Sparse Merkle Tree

***Abstract*—Growing of privacy concerns, Self-Sovereign Identity (SSI), which allows owners to directly control their identity information, is gaining prominence. Currently, Hyperledger Indy and Aries are being used to implement SSI systems, enabling the issuance, verification, and revocation of credentials. The credential revocation system based on Hyperledger Indy operates through cryptographic accumulators and associated tails files. However, this approach has several drawbacks: the setup process for tails files is time-consuming, credential revocation cannot be performed if the server storing the tails files is inaccessible, and there is a limit on the number of credentials that a single tails file can handle. This paper addresses these issues by proposing a replacement of tails files with zk-SNARK and Sparse Merkle Tree, offering a more efficient and scalable solution. We have implemented a prototype of our system and evaluated in terms of revocation efficiency.**

***Index Terms*—zk-SNARK, Sparse Merkle Tree, Credential, Revocation**

I. Introduction

Self-Sovereign Identity (SSI) [1]–[3] represents a decentralized digital identity system that empowers individuals to own and control their digital identity information, enabling them to manage and share their identity data without the need for central authorities or intermediaries. Unlike traditional identity verification systems where identity information is stored in centralized databases, SSI poses a lower risk of data breaches [4], [5]. Additionally, it leverages distributed ledger technologies, such as blockchain, to provide transparency and reliability of data.

The Hyperledger Foundation is currently implementing SSI systems through the Hyperledger Indy [6] and Hyperledger Aries [7] projects. Hyperledger Indy deals with distributed ledger technology for managing data, while Hyperledger Aries implements communication between the distributed ledger and clients through the Aries Cloud Agent (ACA) [8]. Fig. 1 illustrates the overall ecosystem of SSI, comprising Issuer, Holder, Verifier, and Distributed Ledger components. The issuer issues a Verifiable Credential (VC) [9], [10] to the holder, which can authenticate the user's identity, and this issuance record is stored on the distributed ledger. The verifier can verify the validity of the VC provided by holder.

Given that a VC contains sensitive information used to prove a user's identity, it is essential to include a revocation process for invalid or expired VCs in the VC lifecycle. For instance, if a VC represents a user's driver's license, it must be necessarily revoked of when the license expires. In Hyperledger Indy, the process of VC revocation is of great significance, as verifiers check the revocation status of a submitted VC when validating its authenticity.

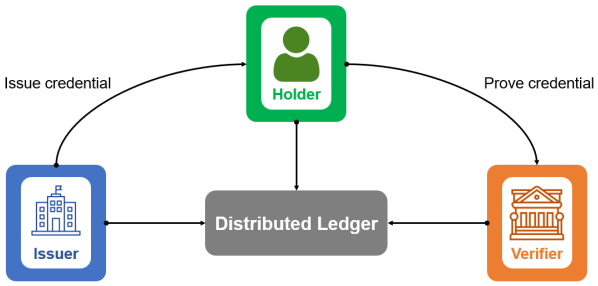

Fig. 1. Self-Sovereign Identity System

In the current Hyperledger Indy framework, the revocation of Verifiable Credential (VC) is based on a cryptographic accumulator [11], which accumulates many individual values from a tails file [12] containing random values corresponding to a large number of credentials, into a single fixed-size value. However, this approach has several issues.

- **Setup Process Overhead**: The creation of the tails file is time-consuming, and the process of registering the generated accumulator on Indy's ledger adds to the overhead in the setup phase.
- **Dependency on Tails File for Credential Revocation**: When a specific credential is revoked, the corresponding value in the tails file is deleted, altering the accumulated value. This change necessitates holders to access the updated tails file to update necessary values, such as the witness for future non-revocation proof generation. If the centralized file server storing the tails file encounters issues, preventing access, holders cannot update their witness values and thus cannot generate correct proofs.
- **Limitations in ACA Implementation**: The Aries-Cloud-Agent (ACA) currently limits the tails file to a maximum of 32,768 values [13]. If the number of issued credentials exceeds this limit, a new tails file is required, necessitating a repeat of the setup process.

In this paper, we propose an alternative approach that utilizing zero-knowledge Succinct Non-interactive Arguments of Knowledge (zk-SNARK) [14]–[16] technology and Sparse Merkle Tree (SMT) [17]–[19], instead of relying on tails file and cryptographic accumulator. **Our objective is to reduce the overhead in the setup process and to solve the centralization and quantity limitation issues by storing the hash of transactions that contains zk-SNARK proof into SMT.**

II. Related Work

After the method of using accumulators for the revocation of credentials was proposed in [20], accumulators have continued to evolve as the most effective solution for revocation. Structures based on accumulators essentially aggregate many credentials to manage them as one final value. Whenever a specific credential is to be revoked, it is excluded from the list, and then a new final value is aggregated again. During this process, witnesses are generated to prove that other credentials have not been revoked [21]. Each time the final value is created, the witness must also be updated.

Various types of accumulators have been researched for efficient revocation of credentials based on accumulator technology. The first is the static accumulator based on the RSA assumption, initially proposed in [22]. It utilized a simple one-way hash function to satisfy the quasi-commutative property. [23] implemented a dynamic accumulator, where components could be dynamically added or removed. Furthermore, ongoing research aims to improve the efficiency of dynamic accumulators based on the RSA assumption [24]–[26].

The second type is the Bilinear-map accumulator, which operates based on elliptic curve groups and the Diffie-Hellman assumption. It was first proposed in [27]. This accumulator has the advantage of having short signature sizes, utilizing ID-based ring signature schemes and group signature schemes. [28] extended this bilinear-map accumulator to support non-membership proofs. This means providing cryptographic proof that a certain element has not been accumulated in a given set, thereby enhancing the performance of the bilinear-map accumulator to a level similar to RSA accumulators. Additionally, research has been conducted on a lightweight version of the bilinear-map accumulator to increase its efficiency [29].

All previous studies share a commonality in that they manage credentials based on accumulators. However, the use of accumulators necessitates a centralized server for managing the credential list in form of files, accessible to all holders or verifiers. A major drawback of this approach is that if the server becomes non-operational, both revocation and verification of credentials become impossible. This problem stems from the use of accumulators. Therefore, we aims to replace cryptographic accumulators using zk-SNARK and sparse merkle trees.

III. Preliminaries

A. zk-SNARK

Zero-knowledge Succinct Non-Interactive Arguments of Knowledge (zk-SNARK) is a specific technique of zero-knowledge proof which is the method of proving a statement is true without revealing any information of statement [30]. zk-SNARK is distinguished by its significantly succinct proof size compared to traditional zero-knowledge proof systems. A key advantage of zk-SNARK is the reduction in communication costs, achieved by eliminating the need for multiple interactions between the prover and verifier through Fiat-Shamir heuristic [31]. Fundamentally, zk-SNARK is composed of three polynomial-time algorithms (KeyGen, Prove, Verify).

- $(pk, vk) \leftarrow KeyGen(1^\lambda, C)$: Generate a proving key pk and verifying key vk using security parameter λ and circuit C.
- $\pi \leftarrow Prove(pk, x, w)$: Generate a proof with proving key pk, public input x, and private witness w.
- $True/False \leftarrow Verify(vk, x, \pi)$: Determine the validity of proof with verifying key vk, public input x, and proof π.

B. Sparse Merkle Tree

Merkle tree is a binary tree structure where each leaf node contains a cryptographic hash value. The parent node is determined by hashing the values of leaf nodes, and this process is repeated up to the root of the tree, thereby forming the complete tree structure. The merkle tree allow efficient management of numerous transactions with low data storage, so it's extensively used in managing blockchain transactions. Additionally, they enable the verification of the inclusion of a specific transaction within a block, which is why they are widely adopted in many blockchain systems [32].

The sparse merkle tree is an expanded structure of the traditional merkle tree. While it maintains the same structure of determining the hash of the parent node from the hash values of the leaf nodes, it differs in that some leaf nodes are empty and do not hold hash values. The tree's depth can be predetermined, allowing for the creation of leaf nodes up to that depth. When a unique hash value is assigned to a specific leaf node, only that leaf node is filled, generating a root value, while the remaining nodes remain empty.

This structure enables SMT to not only generate inclusion proofs, verifying the inclusion of a specific value, but also to create non-inclusion proofs. These non-inclusion proofs demonstrate the absence of a specific value, a feature not available in traditional merkle tree. This dual capability of proving both inclusion and non-inclusion is a significant advantage of SMT.

IV. Methodology

In this section, we describe the overall design of our solution. The overall structure of the system is depicted in Fig. 2. This structure maintains the SSI architecture based on Hyperledger Indy and Hyperledger Aries, as seen in Fig. 1, with the addition of a revocation management system based on zk-SNARK and SMT (Sparse Merkle Tree).

A. Proof Generation Module

In the Indy Self-Sovereign Identity (SSI) System, a holder requests the issuance of a VC from an issuer. The issuer, based on the information submitted by the holder, proceeds with the issuance of the VC. During this process, the proof generation module receives information regarding the validity of the VC from the issuer.

The validity information of the VC used in this system encompasses two main aspects. Firstly, it verifies whether the holder's request conforms to the credential definition of the VC. The credential definition, which outlines the structure

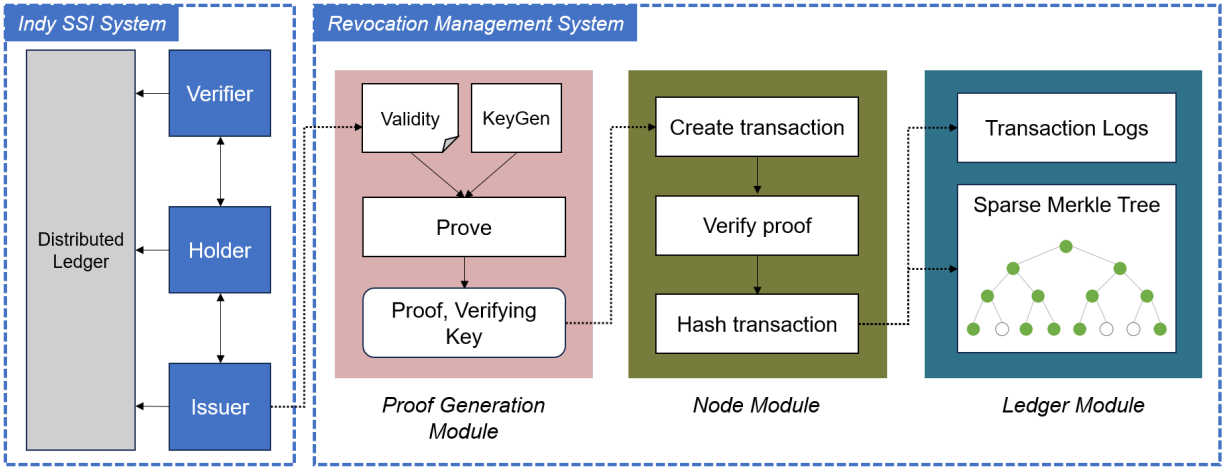

Fig. 2. System Overview

and characteristics of the VC, not only includes elements of a predefined schema but also encompasses the issuer's private key-based signature and public key. Verifying the conformity to the VC's credential definition ensures the VC's validity. Secondly, if the schema of the VC includes an expiration date, it's essential to verify that the VC's validity period is set beyond the current time. This ensures that the VC is not only structurally valid but also temporally relevant.

These validity information is used as an input in the proof generation module. A zk-SNARK proof is generated using the proving key, created through the *KeyGen* algorithm, and the credential validity information received from the issuer as private inputs. By utilizing zk-SNARKs, the holder can prove the validity of the VC without revealing their personal information provided to the issuer in accordance with the credential definition, as well as the expiration information of the VC. Finally, the generated proof, along with the verifying key produced through the *KeyGen* algorithm, is forwarded to the node module. This process ensures that the validity of the VC is authenticated while maintaining the privacy of the holder's sensitive information.

B. Node Module

The node module operates in three stages. First, it receives the proof and verifying key generated by the proof generation module and creates a transaction that includes them. In the prototype of this system, the process of propagating and verifying this transaction across various nodes is omitted, but typically, nodes create transactions and broadcast them to other nodes for verification. The nodes that receive this transaction perform validity checks using the proof and verifying key contained within the transaction.

Once the transaction's validity is confirmed, the final step is to hash the transaction and forward it to the ledger module. This process ensures that the transaction, validated for its authenticity and integrity, is securely and efficiently recorded in the system's ledger, maintaining the robustness and reliability of the overall system.

C. Ledger Module

The ledger module consists of a ledger that stores transaction logs to prevent tampering with transactions received from the node module, and a Sparse Merkle Tree (SMT) that stores the hashes of these transactions. By storing the transaction hashes in the SMT, it becomes possible to generate an inclusion proof that demonstrates a transaction is part of the tree, and a non-inclusion proof that evidences a transaction is not included in the tree.

In the Indy Self-Sovereign Identity (SSI) system, when a verifier receives a Verifiable Credential (VC) from a holder, the ledger module's SMT can be used to check whether that VC has been revoked. The VC includes the transaction hash generated in the node module, allowing for a search in the SMT using this hash value. If the hash is found in the SMT, it proves that the VC has not been revoked; if not, it indicates that the VC has been revoked.

The revocation of a VC is carried out by the issuer of the Indy SSI system. The issuer identifies the transaction hash stored in the VC and deletes the corresponding leaf node value in the SMT that holds this hash. Once this deletion is complete, the VC is effectively removed from the SMT. Subsequently, a verifier can easily confirm the revocation by accessing the SMT and utilizing the non-inclusion proof. This mechanism ensures a secure and verifiable process for managing the validity and revocation of credentials within the SSI framework.

V. Evaluation

A. Experiment Setup

We have implemented a prototype [33] of this system as well as an SSI system. The prototype was developed in the Rust, utilizing the Bellman [34] for zk-SNARK implementation and the Monotree [35] for SMT implementation. The SSI system was implemented with the ACA-py [36] framework to establish the issuer, holder, and verifier structure, and the Von-network [37] was used to implement a distributed ledger.

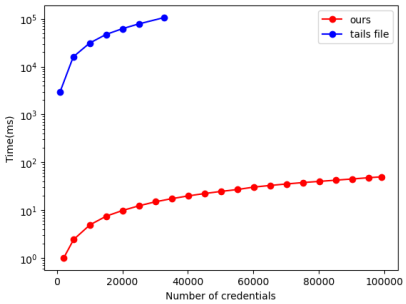

Fig. 3. Setup time for tails file or SMT

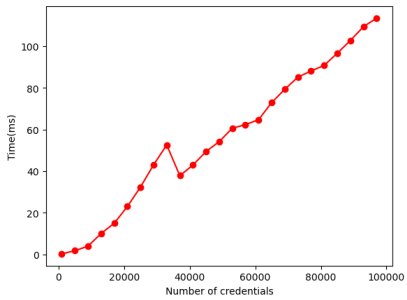

Fig. 4. Revocation speed for SMT

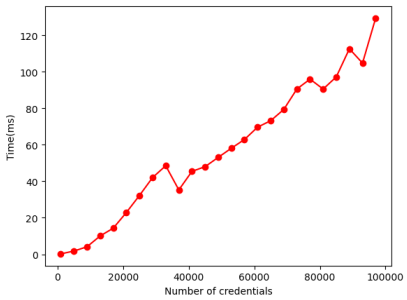

Fig. 5. Non-inclusion proof for SMT

All experiments were conducted on a platform running Ubuntu 18.04, equipped with a 2.85GHz AMD EPYC 7443P processor and 64GB RAM.

B. Performance Evaluation

In our study, we defined the setup process for the SSI method as the time taken to generate tails files. For our approach, it was defined as the time required to create a SMT corresponding to the number of credentials. In Fig. 3, we observed that the time to create an SMT is significantly less than that for generating tails files. For instance, while it took 62s to generate a tails file for 20,000 credentials, it only took 7.5ms to create an SMT, which is more than 8000 times faster. Moreover, in the conventional ACA-py implementation, a single tails file can manage a maximum of 32,768 credentials. In contrast, our method can create an SMT managing over 100,000 credentials in significantly less time than it takes to generate a tails file.

We also evaluated the time taken to generate and verify zk-SNARK proofs for credential validity. The constraints used in the validity proof are determined by the credential definition. As the credential definition becomes more complex, the amount of information to be verified increases, leading to more constraints and, consequently, longer proof generation times. However, in our prototype, a predefined credential definition was used which consists of name, email, and expiration date. We found that the proof generation time was 1.3s, and the verification time was 331.1ms. Even when considering the time for the proof to be propagated and verified by other nodes, these results are sufficiently practical.

Last, we evaluated the speed of revoking credentials and the execution time of generating non-inclusion proofs to confirm that a credential has been revoked. In our system, revoking a credential involves finding and deleting its corresponding transaction hash in the SMT, which was generated at the time of issuance. Fig. 4 demonstrates that as the number of credentials forming the SMT increases, the time to search a specific hash value also increases almost linearly. For an SMT composed of 85,000 leaf nodes, the time taken to find and delete a particular hash value is only about 0.1s, indicating that credentials can be revoked quickly.

After a credential is revoked, when a holder submits this credential to a verifier in the SSI system, the verifier can check the revocation status using a non-inclusion proof. Fig. 5 shows that the time to generate a non-inclusion proof also increases linearly as the number of credentials in the SMT grows. Similar to the revocation process, in an SMT with 85,000 leaf nodes, the time to prove the absence of a specific hash value is approximately 0.11s, demonstrating that non-inclusion proofs can be generated fast.

C. Discussion

There are two limitiations of our implementation. First, we only measured the speed of our method and not compared to Hyperledger SSI method. We even know that the speed of SMT is practical, but we have to evaluate the speed of revocation and non-inclusion proof provided by Hyperledger. It needs to setup the different limitation to the number of credentials for each credential definition, and we need to evaluate the speed of revocation when credentials are issued to the limit. Second, our prototype doesn't implement the transaction propagation between the real node, so we didn't include the transaction propagation time in setup time. However, we can think that even the propagation time is included to setup time, it still shows better performance than tails file.

VI. Conclusion & Future Work

In this paper, we address and resolve the problems arising from the use of cryptographic accumulators and tails files for VC revocation in SSI systems, employing zk-SNARK and Sparse Merkle Tree. By eliminating the need for tails files, we addressed the issue of significant time consumption during the setup process inherent in traditional methods. We also resolved the limitation on the number of tails files in the existing Hyperledger Indy system. Furthermore, the use of zk-SNARK enabled the validation of VC's authenticity without disclosing its information. Also, the rapid speed for node deletion and search in SMT significantly enhanced the speed of revocation during the setup process and non-inclusion proof.

In the future, we will apply our system to the Hyperledger Indy system using plugin, allowing access by multiple distributed nodes and resolving the dependency issues with tails file.

REFERENCES

[1] A. Mühle, A. Grüner, T. Gayvoronskaya, and C. Meinel, "A survey on essential components of a self-sovereign identity," *Computer Science Review*, vol. 30, pp. 80–86, 2018.

[2] P. J. Windley, "Sovrin: An identity metasystem for self-sovereign identity," *Frontiers in Blockchain*, vol. 4, p. 626726, 2021.

[3] F. Wang and P. De Filippi, "Self-sovereign identity in a globalized world: Credentials-based identity systems as a driver for economic inclusion," *Frontiers in Blockchain*, vol. 2, p. 28, 2020.

[4] D. Van Bokkem, R. Hageman, G. Koning, L. Nguyen, and N. Zarin, "Self-sovereign identity solutions: The necessity of blockchain technology," *arXiv preprint arXiv:1904.12816*, 2019.

[5] A. Mühle, A. Grüner, T. Gayvoronskaya, and C. Meinel, "A survey on essential components of a self-sovereign identity," *Computer Science Review*, vol. 30, pp. 80–86, 2018.

[6] M. P. Bhattacharya, P. Zavarsky, and S. Butakov, "Enhancing the security and privacy of self-sovereign identities on hyperledger indy blockchain," in *2020 International Symposium on Networks, Computers and Communications (ISNCC)*. IEEE, Conference Proceedings, pp. 1–7.

[7] "Hyperledger aries," https://www.hyperledger.org/projects/aries, Hyperledger Foundation, accessed: 2023-12-14.

[8] Hyperledger, "Hyperledger aries cloud agent python," https://github.com/hyperledger/aries-cloudagent-python, 2023, accessed: 2023-12-14.

[9] J. Sedlmeir, R. Smethurst, A. Rieger, and G. Fridgen, "Digital identities and verifiable credentials," *Business Information Systems Engineering*, vol. 63, no. 5, pp. 603–613, 2021.

[10] R. Mukta, J. Martens, H.-y. Paik, Q. Lu, and S. S. Kanhere, "Blockchain-based verifiable credential sharing with selective disclosure," in *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. IEEE, Conference Proceedings, pp. 959–966.

[11] J. Camenisch, M. Kohlweiss, and C. Soriente, "An accumulator based on bilinear maps and efficient revocation for anonymous credentials," in *Public Key Cryptography–PKC 2009: 12th International Conference on Practice and Theory in Public Key Cryptography, Irvine, CA, USA, March 18-20, 2009. Proceedings 12*. Springer, Conference Proceedings, pp. 481–500.

[12] D. Hardman, "0011: Credential revocation - hyperledger indy hipe," https://hyperledger-indy.readthedocs.io/projects/hipe/en/latest/text/0011-cred-revocation/README.html, 2018, accessed: 2023-12-14.

[13] B. C. Government, "Indy tails server," https://github.com/bcgov/indy-tails-server, 2023, accessed: 2023-12-14.

[14] T. Chen, H. Lu, T. Kunpittaya, and A. Luo, "A review of zk-snarks," *arXiv preprint arXiv:2202.06877*, 2022.

[15] A. Ozdemir and D. Boneh, "Experimenting with collaborative zk-SNARKs:Zero-Knowledge proofs for distributed secrets," in *31st USENIX Security Symposium (USENIX Security 22)*, Conference Proceedings, pp. 4291–4308.

[16] M. Petkus, "Why and how zk-snark works," *arXiv preprint arXiv:1906.07221*, 2019.

[17] R. Dahlberg, T. Pulls, and R. Peeters, "Efficient sparse merkle trees: Caching strategies and secure (non-) membership proofs," in *Secure IT Systems: 21st Nordic Conference, NordSec 2016, Oulu, Finland, November 2-4, 2016. Proceedings 21*. Springer, Conference Proceedings, pp. 199–215.

[18] B. Ma, V. N. Pathak, L. Liu, and S. Ruj, "One-phase batch update on sparse merkle trees for rollups," *arXiv preprint arXiv:2310.13328*, 2023.

[19] Z. Gao, Y. Hu, and Q. Wu, "Jellyfish merkle tree," 2021.

[20] J. Camenisch and A. Lysyanskaya, "Dynamic accumulators and application to efficient revocation of anonymous credentials," in *Advances in Cryptology—CRYPTO 2002: 22nd Annual International Cryptology Conference Santa Barbara, California, USA, August 18–22, 2002 Proceedings 22*. Springer, 2002, pp. 61–76.

[21] N. Fazio and A. Nicolisi, "Cryptographic accumulators: Definitions, constructions and applications, 2003," *URL https://cs. nyu. edu/ fazio/research/publications/accumula tors. pdf*.

[22] J. Benaloh and M. De Mare, "One-way accumulators: A decentralized alternative to digital signatures," in *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer, Conference Proceedings, pp. 274–285.

[23] J. Li, N. Li, and R. Xue, "Universal accumulators with efficient non-membership proofs," in *Applied Cryptography and Network Security: 5th International Conference, ACNS 2007, Zhuhai, China, June 5-8, 2007. Proceedings 5*. Springer, Conference Proceedings, pp. 253–269.

[24] F. Baldimtsi, J. Camenisch, M. Dubovitskaya, A. Lysyanskaya, L. Reyzin, K. Samelin, and S. Yakoubov, "Accumulators with applications to anonymity-preserving revocation," in *2017 IEEE European Symposium on Security and Privacy (EuroSP)*. IEEE, Conference Proceedings, pp. 301–315.

[25] J. Camenisch and A. Lysyanskaya, "Dynamic accumulators and application to efficient revocation of anonymous credentials," in *Advances in Cryptology—CRYPTO 2002: 22nd Annual International Cryptology Conference Santa Barbara, California, USA, August 18–22, 2002 Proceedings 22*. Springer, Conference Proceedings, pp. 61–76.

[26] A. Mashatan and S. Vaudenay, "A fully dynamic universal accumulator," *Proceedings Of The Romanian Academy Series A-Mathematics Physics Technical Sciences Information Science*, vol. 14, no. ARTICLE, pp. 269–285, 2013.

[27] L. Nguyen, "Accumulators from bilinear pairings and applications to id-based ring signatures and group membership revocation," *Topics in Cryptology-CT-RSA 2005*, pp. 275–292.

[28] I. Damgård and N. Triandopoulos, "Supporting non-membership proofs with bilinear-map accumulators," *Cryptology ePrint Archive*, 2008.

[29] Y. Yang, H. Cai, Z. Wei, H. Lu, and K.-K. R. Choo, "Towards lightweight anonymous entity authentication for iot applications," in *Information Security and Privacy: 21st Australasian Conference, ACISP 2016, Melbourne, VIC, Australia, July 4-6, 2016, Proceedings, Part I 21*. Springer, Conference Proceedings, pp. 265–280.

[30] M. Petkus, "Why and how zk-snark works," *arXiv preprint arXiv:1906.07221*, 2019.

[31] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Conference on the theory and application of cryptographic techniques*. Springer, Conference Proceedings, pp. 186–194.

[32] H. Sheth and J. Dattani, "Overview of blockchain technology," *Asian Journal For Convergence In Technology (AJCT) ISSN-2350-1146*, 2019.

[33] alterkim, "Credential revocation with zk-snark and sparse merkle tree," https://github.com/alterkim/smt-revocation, 2023, accessed: 2023-12-18.

[34] zkcrypto, "Bellman: zk-snark library," https://github.com/zkcrypto/bellman, 2023, accessed: 2023-12-14.

[35] thyeem, "Monotree: An optimized sparse merkle tree in rust," https://github.com/thyeem/monotree, 2023, accessed: 2023-12-14.

[36] Hyperledger, "Hyperledger aries cloud agent python," https://github.com/hyperledger/aries-cloudagent-python, 2023, accessed: 2023-12-14.

[37] B. C. Government, "Von network: A portable development level indy node network," https://github.com/bcgov/von-network, 2023, accessed: 2023-12-14.