Zero-Knowledge Shuffle Improvement in Ethereum Single Secret Leader Election

Anders Malta Jakobsen*, Oliver Holmgaard†

Abstract—This is the abstract Zero-Knowledge Proof (ZKP) [1].

Index Terms—Ethereum, Proof of Shuffle, Distributed Systems, Inner Product Arguments, Zero-Knowledge Proof

1 Introduction

This is the introduction

Related Work

This is related work

2 Related Work

2.1 Single Secret Leader Election

2.2 Shuffling algorithm

The shuffling algorithm used in curdleproofs has gone though many iterations and improvements in order to increase speed and reduce the size the proof. This is because the proposer has a limited amount of time to propose a block in each slot, and the addition of the proof to the protocol increases the size of the block the proposers have to create. This is the reason why the current implementation of curdleproofs has chosen the shuffling algorithm [2] proposed by Larsen et al.

The way the shuffle works is by selecting 2 days' worth of proposers, and then shuffling the proposers over one day's worth of slots to create a new list of proposers for the following day. In each slot a subset of the proposers are shuffled, and the rest are left unchanged.

Though experiments Larsen et al. has shown that after enough shuffles becomes secrue even in adversarial environments. They also surgests that their may be room to lower the size of the subsets chosen in each lot without losing the security of the shuffle. Thereby increasing the speed of the shuffle and reducing the size of the proof being added to the blockchain.

- *All authors are affiliated with the Dept. of Computer Science, Aalborg University, Aalborg, Denmark*
- *E-mails: *amja23, †oholmg20 @student.aau.dk*

2.3 Bulletproofs

A big inspiration for the curdleproofs protocol is the use of bulletproofs [3]. Bulletproofs is a type of range proof that uses inner product arguments to prove that a committed value is within a certain range without revealing the value itself. Bulletproofs is in itself not a zero-knowledge proof system, but with the help of Fiat Shamir [3] it can be used to create a zero-knowledge proof. Bulletproofs also has had a few iterations and improvements to increase the speed and reduce the size of the proof since it was used in curdleproofs. One of these is Bulletproofs+ [4] which is a new version of bulletproof that uses a weighted inner product argument instead of the standard inner product argument to achieve a better performance. Bulletproofs+ is also different because it is zero-knowledge proof by itself unlike the original bulletproofs. A third version of the bulletproofs is Bulletproofs++ [5] which is a even newer version of bulletproofs that uses a new type of argument called the norm argument to achieve a better performance. Unlike the two other proofs Bulletproofs++ is a binary range proof, which means that even if it is the fastest proof it is not suitable for the curdleproofs protocol due to the binary nature of the bulletproofs++.

3 Background

3.1 Notation

The notation used throughout this paper can be seen in Table 1.

3.2 Security Assumptions

Since this work is based on the existing Curdleproofs protocol [6], it inherits the same security assumptions. Our work therefore runs as a public coin protocol in any cryptographic group where Decisional Diffie-Hellman (DDH) is hard [7].

3.3 Whisk

3.3.1 Ethereum

Ethereum is a decentralized blockchain platform that enables developers to build and deploy smart contracts and decentralized applications. It is the second-largest blockchain platform by market capitalization and has a large and active developer community. Ethereum uses a proof-of-stake consensus mechanism, which allows users to validate

Symbol	Description
$\mathbb{G}$	Cyclic, additive, group of prime order p
$\mathbb{Z}_p$	Ring of integers modulo p
$\mathbb{G}^n,\ \mathbb{Z}_p^n$	Vector spaces of dimension n over $\mathbb{G}$ and $\mathbb{Z}_p$
$\mathbb{Z}_p^*$	Multiplicative group $\mathbb{Z}_p \setminus \{0\}$
$H \in \mathbb{G}$	Generator of $\mathbb{G}$
$\gamma \in \mathbb{Z}_p^{\lceil \log n \rceil}$	Uniformly distributed challenges
$\mathbf{a} \in \mathbb{F}^n$	Vector $\mathbf{a} = (a_1, \ldots, a_n) \in \mathbb{F}^n$
$\mathbf{A} \in \mathbb{F}^{n \times m}$	Matrix with n rows and m columns
$\mathbf{b} = c \cdot \mathbf{a} \in \mathbb{Z}_p^n$	The vector where $b_i = c\, a_i$, with scalar $c \in \mathbb{Z}_p$ and $\mathbf{a} \in \mathbb{Z}_p^n$
$\mathbf{a} \times \mathbf{b} = \sum_{i=1}^{n} a_i \cdot b_i$	Inner product of $\mathbf{a}, \mathbf{b} \in \mathbb{F}^n$
$\mathbf{G} = (g_1, \ldots, g_n) \in \mathbb{G}^n, \mathbf{G}' = (g_1', \ldots, g_n') \in \mathbb{G}^n$	Vectors of generators (for Pedersen commitments)
$A = a \times G = \sum_{i=1}^{n} a_i \cdot G_i$	Binding (but not hiding) commitment to $a \in \mathbb{Z}_p^n \in$
$\mathbf{r}_A \in \mathbb{Z}^n$	Blinding factors, e.g. $A = \mathbf{a} \times \mathbf{G} + \mathbf{r}_A \times \mathbf{G}$ is a Pedersen commitment to $\mathbf{a}$
$\mathbf{a} \parallel \mathbf{b} \in \mathbb{Z}_p^{n+m}$	Concatenation: if $\mathbf{a} \in \mathbb{Z}_p^n$, $\mathbf{b} \in \mathbb{Z}_p^m$, then $\mathbf{a} \parallel \mathbf{b} \in \mathbb{Z}_p^{n+m}$
$\mathbf{a}_{[:k]} = (a_1, \ldots, a_k) \in \mathbb{F}^k,\ \mathbf{a}_{[k:]} = (a_{k+1}, \ldots, a_n) \in \mathbb{F}^{n-k}$	Slices of vectors (Python notation)
{*Public Input*, *Witness*}: Relation	Relation using the specified public input and witness

TABLE 1: Notation used throughout the paper.

transactions and create new blocks by staking their Ether (ETH) tokens. The Proof-of stake protocol works in epochs of 32 slots, where each slot is 12 seconds long. In each slot a proposer is chosen to propose a block thereby allowing the network to reach consensus on the state of the blockchain.

3.3.2 Proposer DoS attack

An attack on the Ethereum network that was discovered by Heimbach et al. [8] is the deanonymization attack on validators. In our preliminary work [9], we have shown that the attack still possible to perform on the Ethereum network, and using the attack, a proposer DoS can be preformed. The proposer DoS attack is a type of attack that targets the block proposers making them unable to propose blocks. An adversary can use the proposer DoS attack to prevent a proposer from receiving rewards, gotten from proposing a block, and increase their oen rewards [10]. As a response to the proposer DoS attack, Ethereum has proposed a new protocol called Whisk [11] as an attempt to mitigate the attack.

3.3.3 The Whisk protocol

Whisk is a zero-knowledge Single Secret Leader Election (SSLE) system that uses a zero-knowledge argument called curdleproofs [6] to verify the correctness of a shuffle without revealing the input or output [12] Whisk works by selecting a list of proposers 16384 and shuffling them over 8192 slots (1 day). Then 8192 proposers are selected from the shuffled list to propose blocks for the next 8192 slots while a new list is being shuffled. This way a new list of proposers is created every day. After each shuffle Whisk uses a zero-knowledge proof to prove that the shuffle is correct. This is so that the proposer can prove that they are the correct proposer for the slot without revealing their identity, thereby mitigating the proposer DoS attack because of the identity of the upcoming proposers being hidden now.

3.3.4 Curdleproofs

Curdleproofs is a zero-knowledge proof system that allows a prover to prove the authenticity of a shuffle without revealing how it was shuffled. It does this by using 3 different zero-knowledge proofs with one of them relying on two more zero-knowledge proofs. the first proof is a sameperm proof. The sameperm proof is used to prove a commitment to a specific, but not publicly known, permutation. Sameperm also runs a subroutine to help with the proof, called a grand product argument. The grand product argument is an intermediate step used to construct an inner product argument, which proves the grand product argument. The second proof is a ”same multiscalar” argument. This proves that permuted set of ciphertexts was made by using the permutation that the prover previously committed to. The third proof is a samescalar argument which proves that, given a public input, there exists a scalar, k, such that the commitment of the permuted set is equal to the commitment of the pre-permuted set multiplied by k.

3.4 Zero-knowledge proofs

Curdleproofs is a zero-knowledge proof system, which means that it allows a prover to convince a verifier that they know a secret without revealing the secret itself. within the context of Ethereum it could be the ability to convince someone that a transaction is valid without revealing information about the transaction such as the value of it.

4 APPROACH

As explained in section 3, Curdleproofs makes use of three different proofs. This work focuses on improving the underlying Inner Product Argument (IPA). Especially the running time and proof size of the protocol are of interest. The following is our approach to, how we modified the IPA.

4.1 Springproofs

In Chapter 6 of Curdleproofs [6], they explain the efficiency of the protocol, including also the size of the proof. They specifically mention that the proof has size $18 + 10\log(\ell+4)\mathbb{G}, 7\mathbb{F}$. As the proof size is dependent on the size of the shuffle, ℓ, an interest in the possibility of reducing this parameter arises. The current proposal of curdleproofs only works on shuffles, where the size is a power of 2. The reason is that the underlying proofs, such as the IPA, needs to fold recursively down to 1, by halving the size in every round.

The Springproofs protocol [13] can be used very effectively in this scenario. The theory of Springproofs provides support for IPAs to use vectors of arbitrary length. Using the findings of Springproofs means Curdleproofs could be used on shuffle sizes other than powers of two. As such, they could lower the shuffle size from the current 128 to a size significantly lower, given it is still secure. Seeing the proof size of Curdleproofs being dependent on ℓ means that this modification would greatly help in lowering it.

One of the most notable findings in Springproofs is the usage of their so-called scheme function. This function is used to ensure that the IPA eventually will fold down to a vector of size 1. In a general IPA, Curdleproofs included, if the size of the vectors were not a power of two, the argument would not recursive down to size 1, as they work by halving the vectors every recursive round.

The core concept of the Springproofs scheme function is to split the vectors into sets, T, S before each recursive round of the protocol. Then, the fold for that round is only done on one of the two sets, T, before the other set, S, is appended again at the end of the recursive round.

Springproofs present different scheme functions and prove some of them to be optimal. One of these optimal functions is an optimized version of their *pre-compression method*, which splits the vectors as seen in Figure 1. The computation is for finding the set, T.

input: n, where $n > 0$

$\{n\} \leftarrow n$
$N \leftarrow 2^{\lceil \log n \rceil - 1}$
$i_h \leftarrow \lfloor (2N - n)/2 \rfloor + 1$
$i_t = \lfloor n/2 \rfloor$
if $n \neq N$:
    $\{T\} \leftarrow (i_h : i_t) \cup (N + 1 : n)$
else if $n = N$:
    $\{T\} \leftarrow (1 : n)$
$\{S\} \leftarrow \{n\} - \{T\}$

Fig. 1: Scheme function *f* used in CAAUrdleproofs

This can also visually be seen in Figure 2(b), which is figure 1 of the Springproofs paper [13]. In Figure 2(a) is a scheme function which simply pads the vector to the next power of two before running an IPA. If one wanted to run current IPAss on vector that are not a power of two, this would generally be the easiest way to achieve that. Though, this defeats the attempt of lowering the proof size, as it would now correspond to running an IPA on the size of the next power of two.

It is notable to mention that using the folding as shown in Figure 2(b) results in the second recursive round being a size corresponding to a power of two. This means that the rest of the protocol will run as a general IPA, without the actual need for splitting the vectors, which can also be seen in Figure 1.

With the idea from Springproofs in mind, we have made a modification to the IPA of Curdleproofs We call this modified protocol, CAAUrdleproof. For generality and readability, we show the split of vectors happening every round.

First of all, we have the prover computation, where the proof is constructed. The construction can be seen in Figure 3.

First, we have step 1, which is the setup phase. It is done exactly the same way as in Curdleproofs. To ensure zero-knowledge, two blinding vectors for each commitment are constructed. These are also given the properties, $(\mathbf{r}_C \times \mathbf{d} + \mathbf{r}_D \times \mathbf{c}) = 0$ and $\mathbf{r}_C \times \mathbf{r}_D = 0$, ensuring the completeness of the protocol.

From the public input, hash values α, β are then computed. These are used to ensure the soundness of the protocol.

The two vectors are then blinded and multiplied by the α hash to ensure the zero-knowledge and soundness, as well as $H = \beta H$.

Now, the recursive proof construction begins. As explained, at the start of the recursive round, the `while`-loop, we find the split of the vectors, with $f(n)$ being the function from Figure 1. Then, we find half the length of the T set, as this is the set, we are doing the recursive round on. Equally we split our witness vectors and the group vectors using T and S.

After this, the prover constructs cross commitment elements that are computed on the T set. These are added to the proof, which eventually is available to the verifier. They are also used to construct a hash value, γ_j, in the next step.

This value is used for completing the folding of $\mathbf{c}, \mathbf{d}, \mathbf{G}, \mathbf{G}'$. We do the fold as in the original Curdleproofs protocol, while also appending the elements of S back onto the vectors. The figure shows a concatenation, but it is important to know that the vectors are appended together as shown in Figure 2(b).

At last, n is updated to the length of the concatenated vectors before starting a new round.

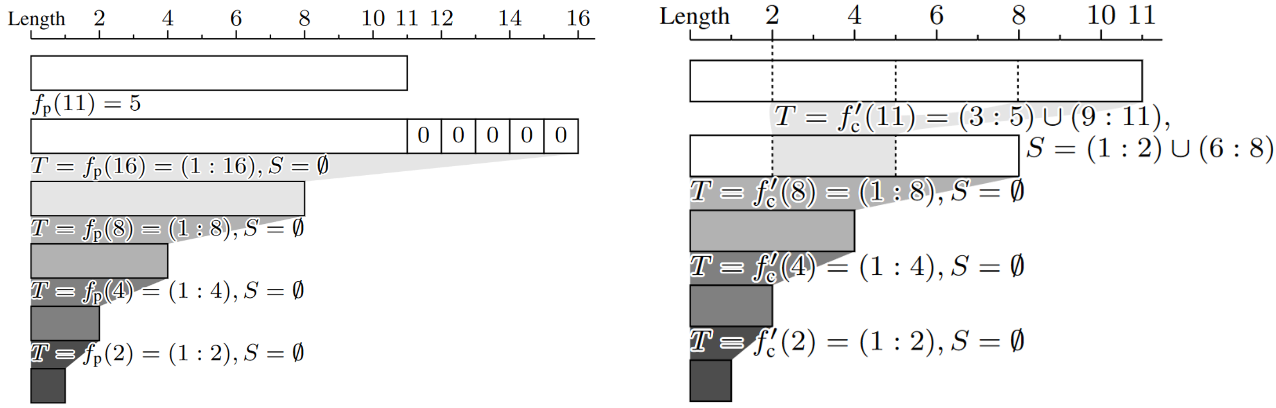
The result of this is a proof constructed in $\lceil \log n \rceil$ rounds, but with the proof size being smaller than if the shuffle size was a power of 2.

The now constructed proof is then supposed to be added to the block in the chain at the given time slot [11]. Having the proof on the blockchain allows for each validator to asynchronously verify whether it is a valid proof. Again, the originally proposed verifying protocol has been modified according to Springproofs, which is seen in Figure 4.

The changes to the verifier protocol are equivalent to the ones made to the prover protocol. First, the vectors are divided into the two sets, $|T|, |S|$. The verifier then retrieves the cross-product commitment update values, $L_{C,j}, L_{D,j}, R_{C,j}, R_{D,j}$. These are used for constructing a new commitment according to the fold made at round i. The corresponding left and right side cross-product are multiplied by a challenge, γ_j, γ_j^{-1}, respectively. By this time, the C and D commitments are a commitment to the original commitments along with the folded commitment.

$\mathbf{G}, \mathbf{G}'$ are updated as in Figure 3 before the protocol updates n to be the length of the newly constructed vectors.

As in the prover protocol, this is then repeated for $\log n$ round, after which the vectors have length 1.



(a) Padding Method

(b) Optimal Pre-Compression Method

Fig. 2: Folding visualization as seen in the Springproofs paper

At the end of the protocol, the verifier now does its final check. From the prover, it has retrieved the folded down c and d vectors. It therefore constructs commitments with these elements. So, it constructs $c \times G_1 + cdH$, which is the structure of the C commitment as well as $d \times G'_1$, which is the structure of the D commitment. The verifier now checks if these commitments match the commitments that he constructed in the recursive protocol. If so, the verifier accepts the proof.

4.2 Shuffle security

The shuffle method proposed by Larsen et al. [2] that was used in curdleproofs is based on the idea of shuffling a list of proposers over a set of slots. The shuffle itself however is not too complex. A formal definition of the shuffle is given in Figure 5.

Here the set $(c_1, \ldots, c_n)$ is a set of ciphertexts that are shuffled over T slots. In each slot t, a subset of the ciphertexts $i_1, \ldots, i_k$ is chosen randomly and shuffled and added back to the list of ciphertexts. It is then encrypts the ciphertexts again and publishes them. This process is repeated for T slots and the shuffle is complete. During the T shuffles some of the shufflers may be adversarial. This means that whenever the shuffling process is taking place a part of the shuffles may be adversarial which can be seen as not being shuffled. Therefore the amount of honest shuffles that happen during the shuffle process is $T_H = T - \alpha$. Where α is the amount of adversarial shufflers.

The Shuffle is secure if none of these two events occur. The first event is a short backtracking, where an adversary can find the original ciphertexts from the shuffled ciphertexts. Since the subsets of ciphertexts are chosen randomly in each shuffle, if is enough adversarial shufflers in a row to end the process, then a shot backtrack is possible.

The second event that can occur is that since every shuffle distributes the possibility of a certain ciphertext to be in a certain slot. Then if a shuffle contains a lot of ciphertext with a larger than average chance of containing a certain ciphertext, then that would imply that there is a higher chance of that ciphertext being in that slot.

It is theoretically possible to find a shuffle size and a number of shuffles given an amount of adversarial shufflers to guarantee that the shuffle is secure. For any $0 < \delta < 1/3$, if $T \geq 20n/k \ln(n/\delta) + \beta$ and $k \geq 256 \ln^2(n/\delta)(1 - \alpha/n)^{-2}$. If T and k are chosen such that the above two conditions are met, then the protocol is an (ϵ, δ)-secure (T, n, k)-shuffle in the presence of a (α, β)-adversary where $\epsilon = 2/(n - \alpha)$.

This formula is the lowest theoretically proven bound for T and k. It is however possible to find lower secure values for T and k but this has to be done experimentally.

4.3 Implementation

5 Experimental Protocol

5.1 Size of curdleproofs

In this experiment we measured the time to run both the original curdleproofs and our version with the addition of springproofs. Curdleproofs was run with the shuffle sizes of 64 and 128 as it has to run with a power of 2.

Since our version has the addition of springproofs, we were able to run it without having to consider the shuffle size being a power of 2. Therefore, our version was run with the shuffle sizes from 64 to 128 while measuring the time it takes to run.

5.2 Shuffle security

In this experiment we ran the shuffle protocol with varying shuffle sizes and varying number of adversarial shufflers. Since the purpose of this experiment is to find the lowest possible shuffle size that is still secure, it was run with a shuffle size between 64 and 128. Because curdleproofs is ment to be used in an Ethereum setting all the experiments was done with 8192 shuffles, since that is the amount of slots it will be shuffled over in Ethereum.

Every experiment was run 100 times and the average time was taken.

6 Results

These are the results And they are very good

7 Discussion

This is the discussion

Step 1:

$\mathbf{r}_C, \mathbf{r}_D \xleftarrow{\$} \mathbb{F}^n$

where $(\mathbf{r}_C \times \mathbf{d} + \mathbf{r}_D \times \mathbf{c}) = 0$ and $\mathbf{r}_C \times \mathbf{r}_D = 0$

$B_C \leftarrow \mathbf{r}_C \times \mathbf{G}$

$B_D \leftarrow \mathbf{r}_D \times \mathbf{G}'$

$\alpha, \beta \leftarrow \text{Hash}(C, D, z, B_C, B_C)$

$\mathbf{c} \leftarrow \mathbf{r}_C + \alpha\mathbf{c}$

$\mathbf{d} \leftarrow \mathbf{r}_D + \alpha\mathbf{d}$

$H \leftarrow \beta H$

Step 2:

$m \leftarrow n$

$\texttt{while}\ \ 1 \leq j \leq \lceil \log m \rceil$:

$\quad T, S \leftarrow \boldsymbol{f}(n)$

$\quad n \leftarrow \frac{|T|}{2}$

$\quad \mathbf{c} = \mathbf{c}_T\ ,\ \ \mathbf{cS} = \mathbf{c}_S$

$\quad \mathbf{d} = \mathbf{d}_T\ ,\ \ \mathbf{dS} = \mathbf{d}_S$

$\quad \mathbf{G} = \mathbf{G}_T\ ,\ \ \mathbf{GS} = \mathbf{G}_S$

$\quad \mathbf{G}' = \mathbf{G}'_T\ ,\ \ \mathbf{GS}' = \mathbf{G}'_T$

$\quad L_{C,j} \leftarrow \mathbf{c}_{[:n]} \times \mathbf{G}_{[n:]} + (\mathbf{c}_{[:n]} \times \mathbf{d}_{[n:]})H$

$\quad L_{D,j} \leftarrow \mathbf{d}_{[n:]} \times \mathbf{G}'_{[:n]}$

$\quad R_{C,j} \leftarrow \mathbf{c}_{[n:]} \times \mathbf{G}_{[:n]} + (\mathbf{c}_{[n:]} \times \mathbf{d}_{[:n]})H$

$\quad R_{D,j} \leftarrow \mathbf{d}_{[:n]} \times \mathbf{G}'_{[n:]}$

$\quad \pi_j \leftarrow (L_{C,j}, L_{D,j}, R_{C,j}, R_{D,j})$

$\quad \gamma_j \leftarrow Hash(\pi_j)$

$\quad \mathbf{c} \leftarrow \mathbf{cS} \| \mathbf{c}_{[:n]} + \gamma_j^{-1}\mathbf{c}_{[n:]}$

$\quad \mathbf{d} \leftarrow \mathbf{dS} \| \mathbf{d}_{[:n]} + \gamma_j\mathbf{d}_{[n:]}$

$\quad \mathbf{G} \leftarrow \mathbf{GS} \| \mathbf{G}_{[:n]} + \gamma_j\mathbf{G}_{[n:]}$

$\quad \mathbf{G}' \leftarrow \mathbf{GS}' \| \mathbf{G}'_{[:n]} + \gamma_j^{-1}\mathbf{G}'_{[n:]}$

$\quad n \leftarrow len(c)$

Step 3:

$c \leftarrow c_1$

$d \leftarrow d_1$

$\texttt{return}\ \ (B_C, B_D, \pi, c, d)$

Fig. 3: Prover computation for CAAU-IPA in CAAUrdleproofs

Step 1:

$(\mathbf{G}, \mathbf{G}', H) \leftarrow \texttt{parse}(crs_{dl_{inner}})$

$(C, D, z) \leftarrow \texttt{parse}(\phi_{dl_{inner}})$

$(B_C, B_D, \pi, c, d) \leftarrow \texttt{parse}(\pi_{dl_{inner}})$

$\alpha, \beta \leftarrow \text{Hash}(C, D, z, B_C, B_D)$

$H \leftarrow \beta H$

$C \leftarrow B_C + \alpha C + (\alpha^2 z)H$

$D \leftarrow B_D + \alpha D$

Step 2:

$m \leftarrow \lceil \log n \rceil$

$\texttt{for}\ \ 1 \leq j \leq m$

$\quad T, S \leftarrow \boldsymbol{f}(n)$

$\quad n \leftarrow \frac{|T|}{2}$

$\quad \mathbf{G} = \mathbf{G}_T\ ,\ \ \mathbf{GS} = \mathbf{G}_S$

$\quad \mathbf{G}' = \mathbf{G}'_T\ ,\ \ \mathbf{GS}' = \mathbf{G}'_T$

$\quad (L_{C,j}, L_{D,j}, R_{C,j}, R_{D,j}) \leftarrow \texttt{parse}(\pi_j)$

$\quad \gamma_j \leftarrow \text{Hash}(\pi_j)$

$\quad C \leftarrow \gamma_j L_{C,j} + C + \gamma_j^{-1} R_{C,j}$

$\quad D \leftarrow \gamma_j L_{D,j} + D + \gamma_j^{-1} R_{D,j}$

$\quad \mathbf{G} \leftarrow \mathbf{GS} \| \mathbf{G}_{[:n]} + \gamma_j\mathbf{G}_{[n:]}$

$\quad \mathbf{G}' \leftarrow \mathbf{GS}' \| \mathbf{G}'_{[:n]} + \gamma_j^{-1}\mathbf{G}'_{[n:]}$

$\quad n \leftarrow \text{len}(\mathbf{G})$

Step 3:

Check $C = c \times G_1 + cdH$

Check $D = d \times G'_1$

return 1 if both checks pass, else return 0

Fig. 4: Verifier computation for CAAU-IPA in CAAUrdleproofs

8 Conclusion

This is the conclusion

9 Future work

This is the future work.

10 Acknowledgements

We want to express our sincere gratitude to Daniele Dell'Aglio and Michele Albano for their supervision and guidance throughout this thesis.

We also acknowledge the usage of AI tools such as ChatGPT, GitHub Copilot, and Grammarly. These have been used for clarification and implementation purposes.

$$\Pi(c_1, \ldots, c_n)$$

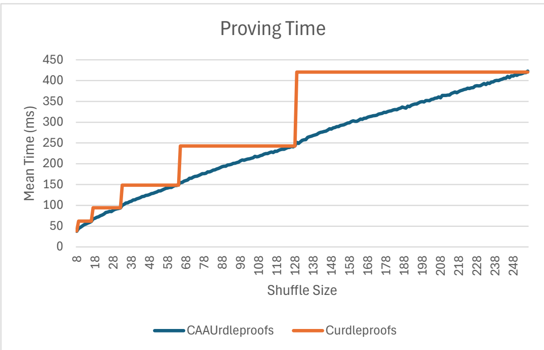
For $t \in [T]$ **:**

S_t picks random $\{i_1, \ldots, i_k\} \subset [n]$

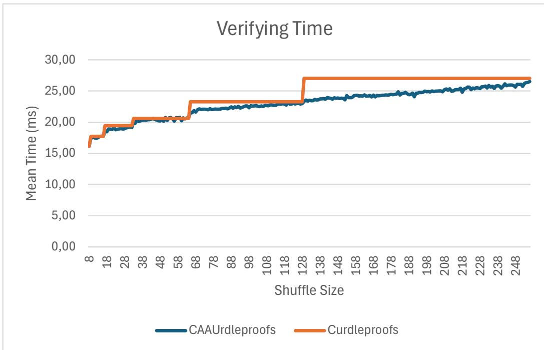
S_t computes $(\tilde{c}_{i_1}, \ldots, \tilde{c}_{i_k}) \leftarrow \text{Shuffle}(c_{i_1}, \ldots, c_{i_k})$

S_t publishes $(\tilde{c}_{i_1}, \ldots, \tilde{c}_{i_k})$

Fig. 5: Distributed shuffling protocol.



(a) Proving Time



(b) Verifying Time

Fig. 6: The timed results compared between CAAUrdleProofs and Curdleproofs

References

[1] G. D. Greenwade, "The Comprehensive Tex Archive Network (CTAN)," *TUGBoat*, vol. 14, no. 3, pp. 342–351, 1993.

[2] K. G. Larsen, M. Obremski, and M. Simkin, *Distributed shuffling in adversarial environments*, Cryptology ePrint Archive, Paper 2022/560, 2022.

[3] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," in *2018 IEEE symposium on security and privacy (SP)*, IEEE, 2018, pp. 315–334.

[4] H. Chung, K. Han, C. Ju, M. Kim, and J. H. Seo, "Bulletproofs+: Shorter proofs for a privacy-enhanced distributed ledger," *Ieee Access*, vol. 10, pp. 42 081–42 096, 2022.

[5] L. Eagen, S. Kanjalkar, T. Ruffing, and J. Nick, "Bulletproofs++: Next generation confidential transactions via reciprocal set membership arguments," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2024, pp. 249–279.

[6] T. E. F. C. R. Team, "Curdleproofs," Accessed: 24-04-2025.

[7] D. Boneh, "The decision diffie-hellman problem," in *Algorithmic Number Theory*, J. P. Buhler, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 48–63, ISBN: 978-3-540-69113-6.

[8] L. Heimbach, Y. Vonlanthen, J. Villacis, L. Kiffer, and R. Wattenhofer, *Deanonymizing ethereum validators: The p2p network has a privacy issue*, 2024. arXiv: 2409.04366 `[cs.CR]`.

[9] A. M. Jakobsen and O. Holmgaard, *"denial of validator anonymity: A de-anonymization attack on ethereum", computer science 9 project*, 2025.

[10] ethereum.org, "Secret leader election," 2024, Accessed: 22-10-2024.

[11] G. Kadianakis, "Whisk: A practical shuffle-based ssle protocol for ethereum," 2024, Accessed: 22-10-2024.

[12] D. Boneh, S. Eskandarian, L. Hanzlik, and N. Greco, "Single secret leader election," in *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, ser. AFT '20, New York, NY, USA: Association for Computing Machinery, 2020, pp. 12–24, ISBN: 9781450381390. DOI: 10.1145/3419614.3423258.

[13] J. Zhang, M. Su, X. Liu, and G. Wang, "Springproofs: Efficient inner product arguments for vectors of arbitrary length," in *2024 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2024, pp. 3147–3164.

Appendix A
Appendix

This is the appendix