Secure Access Control for Files using CL Accumulator and Security analysis of Zerocoin

Aditya Kumar

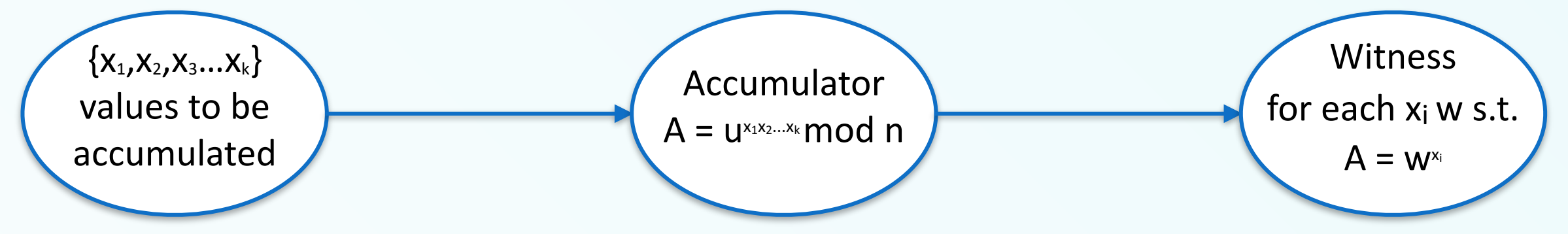
SURGE Intern, Computer Science Department, Indian Institute of Technology Kanpur

Abstract

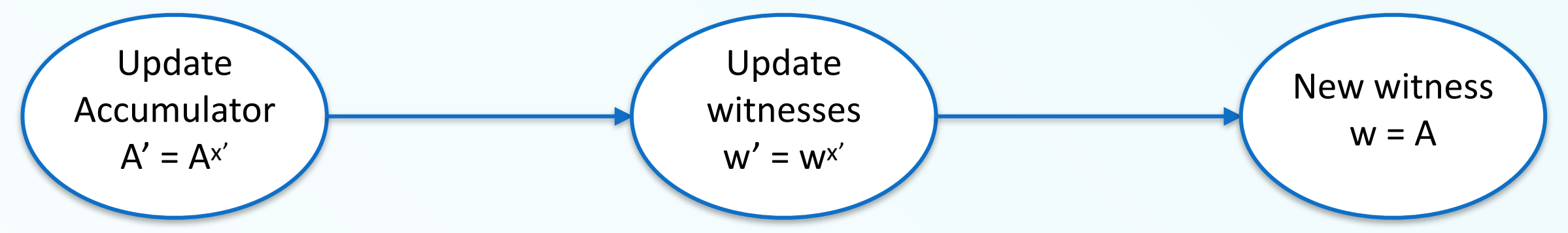
Zcash is one of the well-known cryptocurrencies in today's world. it enables users to pay each other privately, such that the transaction hides the payment's origin, destination and transferred amount. It uses the **zero -knowledge Succinct Non-interactive Arguments of Knowledge (zk-SNARKs).** In our ultimate efforts to study and demystify this zk-SNARK, we studied **Zerocoin**, any early implementation of similar ideas where the payment's origin is supposedly unlinked with the transaction. Still, the destination and amounts are publicly known. They implement this using CL dynamic accumulator. I studied this CL dynamic accumulator and designed a Secure File Access System that uses this accumulator and the Intel SGX as TEE.

Dynamic Accumulator

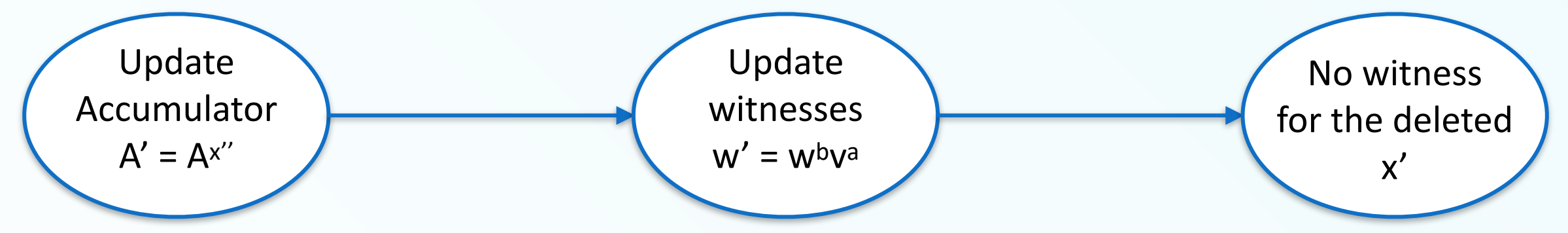
An accumulator scheme is an algorithm that allows one to hash a large set of inputs into one short value, called the accumulator, such that there is a short witness that a given input was incorporated into the accumulator. Finding a witness for a value that was not accumulated should be infeasible. An accumulator must be efficient to generate, evaluate, secure, and quasi-commutative, i.e. the ac- cumulator's value should be the same regardless of the order in which the values were added to the accumulator.
An accumulator that allows dynamic addition and deletion of inputs such that the cost of an add or delete is independent of the number of accumulated values is called a **Dynamic Accumulator**
For applying this Dynamic Accumulator, we require an efficient zero-knowledge proof with which a prover can convince the verifier that he knows a value and its corresponding witness without revealing the witness to the verifier. The **CL Dynamic Accumulator** uses RSA to accomplish this. The starting value u is a quadratic residue modulo n and the accumulated values are all primes between A and B s.t. $B<A^2$.


$\{x_1,x_2,x_3...x_k\}$ values to be accumulated
Accumulator $A = u^{x_1x_2...x_k} \bmod n$
Witness for each x_i w s.t. $A = w^{x_i}$

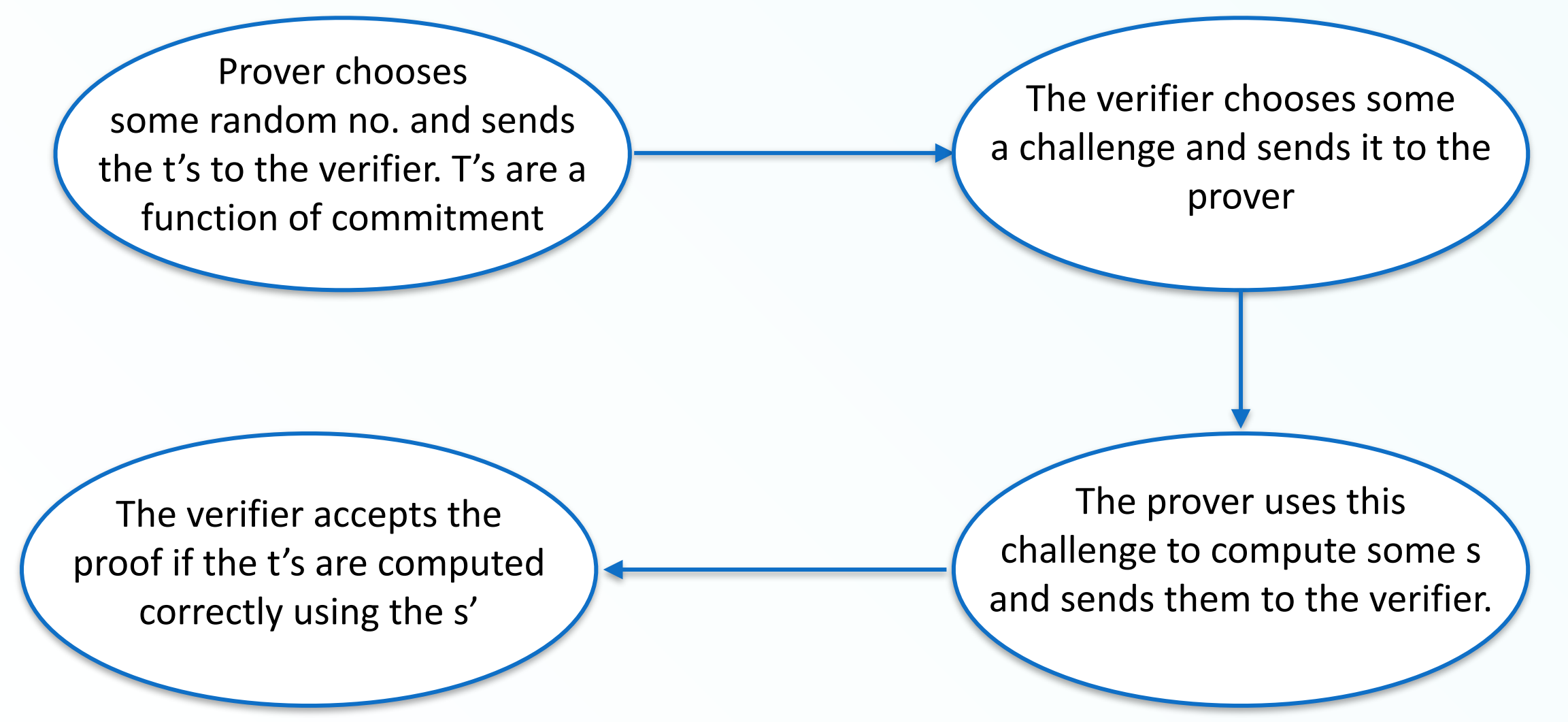
To Add a value x' to the accumulator


Update Accumulator $A' = A^{x'}$
Update witnesses $w' = w^{x'}$
New witness w = A

To Delete a value x' to the accumulator : $x'' = x'^{-1} \bmod (p-1)(q-1)$


Update Accumulator $A' = A^{x''}$
Update witnesses $w' = w^b v^a$
No witness for the deleted x'

$x'' = x'^{-1} \bmod (p-1)(q-1)$ and a, b s.t. $ax + bx' = 1$ x is the value whose witness we are trying to update
There is a zero-knowledge proof by which the prover can prove the knowledge of a witness for a value in the accumulator. To do this, he first commits x by picking a random r Commit(x, r) := $g^x h^r$ where g and h are two generators of a group in which a discrete log problem is hard to solve. The prover and verifier then exchange information so that the prover can convince the verifier that he knows the witness.


Prover chooses some random no. and sends the t's to the verifier. T's are a function of commitment
The verifier chooses some a challenge and sends it to the prover
The prover uses this challenge to compute some s and sends them to the verifier.
The verifier accepts the proof if the t's are computed correctly using the s'

Zerocoin

Zerocoin is a distributed e-cash system that uses cryptographic techniques to break the link between individual Bitcoin transactions without adding trusted parties. It uses a decentralized e-cash scheme based on the dynamic accumulators discussed above.
A decentralized e-cash scheme has the following algorithms

- **Setup** - to set the global parameters to be later used
- **Mint** - to produce a coin c and its associated trapdoor information
- **Spend** - to spend c and produce a transaction consisting of proof π and the serial number S
- **Verify** - to verify a transaction that is to show that proof and S are valid

The proof π is the signature of knowledge for the corresponding zero-knowledge proof for the knowledge of the trapdoor information of one of the coins converted to non-interactive proof using the Fiat-Shamir heuristic.
The Zerocoin protocol requires such a decentralized e-cash scheme to be correct and secure. The security property is based on two characteristics.

- **Anonymity** - The Adversary should not be able to link a given coin spend transaction to the coin associated with it.
- **Balance** - The Adversary should not be able to produce m + 1 spend transactions using only m coins.

The above algorithms are used and added to the Bitcoin blockchain in the following way.

- **Setup(1^λ) → params.** λ is the security parameter. It generates (N,u) used in the accumulators and 2 primes p,q s.t. $p = 2^w q + 1$ for w≥1. Selects two random generators g,h such that the group $< g >=< h >$ is a subgroup of $\mathbf{Z}^*_q$. Output params = (N,u,p,q,g,h). This Setup is run once by a trusted party.
- **Mint (params) → (c,skc).** Select S,r ← $\mathbf{Z}^*_q$ and compute c ← $g^s h^r$ mod p such that c is prime and c ∈ [A,B] (Here, A and B are the ones used in dynamic accumulators). skc = (S,r) and output (c, skc). To mint a Zerocoin c of denomination d, the client (Alice) runs Mint(params) → (c, skc) and stores skc securely. She then embeds c in the output of a Bitcoin transaction that spends d + fees classical bitcoins.
- S**pend (params, c, skc, R, C) → (π, S).** R is some string storing some information about the transaction. Accumulate the set of coins C in the accumulator starting from u and calculate the witness for c using the accumulator value for C – {c}. Output (π, S) where π is the signature of knowledge for the knowledge of (c, w, r) such that c is accumulated in the accumulator, w is its corresponding witness and r such that $c = g^s h^r$. It consists of two zero-knowledge proofs, one for the above dynamic accumulator and the other for the Pederson's Commitment Scheme. To spend c with some other client (Bob), Alice first constructs a partial transaction ptx that references an unclaimed mint transaction and includes Bob's public key as output. She then traverses all valid mint transactions in the blockchain and calls this set of valid coins C. She then runs the above spend algorithm and embeds (π,S) in the scriptSig of the input of ptx. The output of this transaction can be another Zerocoin Mint transaction.
- **Verify(params, π, S, R, C) → {0,1}.** It first accumulates C in u and verifies π for the signature of knowledge on R using the known public values. When the spend transaction appears on the network, the Bitcoin miners run the above algorithm and check that S does not appear in any previous transaction. The block is then added to the blockchain.

Security Analysis of Zerocoin

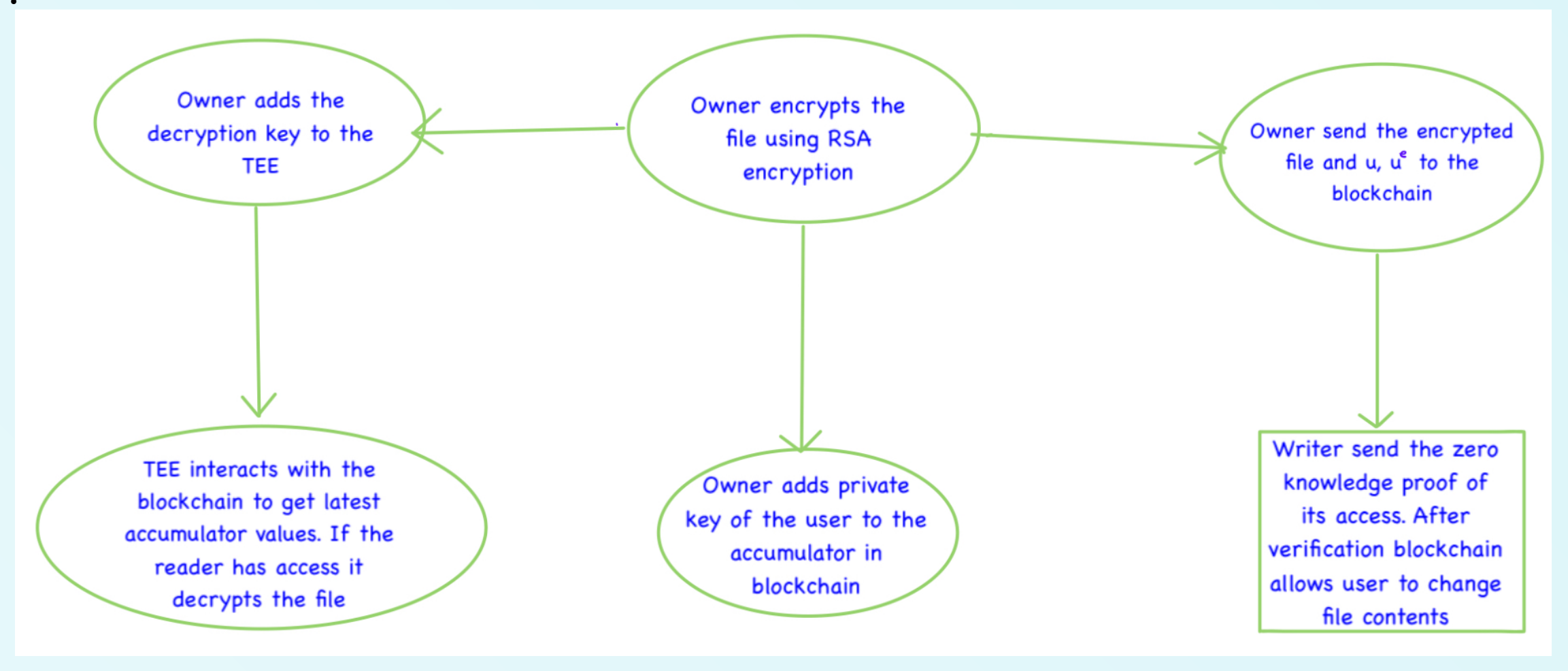
I tried to mount the three attacks to analyse the security of Zerocoin.

- I first tried to use the verify function of the Zerocoin. Verify function tells us if C contains the coin spent in the Spend transaction. So a naive attack was to iterate over all C and check whether Verify returns true. So in this way, the first coin at which Verify returns true is the coin that was spent. Had this attack worked, we could break the Anonymity in logarithmic time, but this doesn't work. The fault lies in the fact that for Verify to return true C must be the same as at the time of the Spend transaction. This means that only one C is valid. So the method of iteration won't work. The Verify asserts that the spender knows a witness to some coin in the current accumulator. It gives no information about any of the previous accumulators.
- I then tried to mount an attack based on the equations of the zero-knowledge proof of the CL dynamic accumulator. I tried various combinations of equations to figure out a function that would return true on inputting a correct witness value pair for a given zero-knowledge proof. (The original paper's authors had proved the correctness of this zero-knowledge but not that it was indeed zero-knowledge if the adversary knew all correct witness value pair.) But all efforts lead us nowhere as every such function I developed returned true even if I paired a witness with some arbitrary random prime. This made our belief that this was indeed zero-knowledge all the more strong.
- We then tried to mount a double spending attack. We minted two coins c_1 and c_2 with secrets (S_1,r_1) and (S_2,r_2) . Now we spend these two coins, and their corresponding commitments to the accumulator are C_{e1} and C_{e2}. Then we tried to generate the ZKP of an imaginary coin with commitment $C_{e1}C_{e2}$ and revealed S_1S_2. The Pederson's Commitment Scheme does hold in this case, and it appears that we might be successful, but the ZKP does not hold in this case. This is because here, by constructing and choosing the parameters A and B of the accumulators carefully, we ensured that $C_{e1}C_{e2} > B$ (as $A^2 > B2^{k'+k''+2}$) this ensures that in the zero- knowledge proof, s_α does not lie in the given range. It is still possible for the ZKP to be valid, but the probability of that happening is quite low. This happens by carefully choosing r_α for the given c and commitments, but since c is itself dependent on r_α, the probability of this happening is a negligible function of the security parameter.

Access Control system

We recognised the potential of the CL Accumulator and focused on its application in access control mechanisms. I successfully designed and implemented a smart contract utilising the CL Accumulator and Intel SGX to grant file access permissions. Here is a brief overview of that system.

- **Create File** - To Add his file to the blockchain, the user first encrypts the File using a private key and sends the encrypted file to the blockchain. Along with this, he also initialises an array of accumulators with two values, the initial u and u^e, where e is his private key. The encrypted file and the accumulator array are then added to the blockchain. The decryption key is stored in Intel SGX, which serves as technology simulating **Trusted Execution Environment(TEE).** (The blockchain part was implemented, and the TEE part was assumed by design).
- **Read Access** - To provide Read Access, the owner sends the decryption key to the reader by providing a way to retrieve the decryption key from TEE. This part was also assumed by design.
- **Grant Write Access** - To grant write access to the user, the owner adds the private key of the user e_1 to the latest accumulator value A. e_1 and A form the valid value witness pair for the user.He appends A^{e_1} to the array of the accumulators in the blockchain. The nodes should first ensure that it is the owner trying to provide access, and to do that, he sends the zero-knowledge proof for the knowledge of the owner's secret key to the blockchain for the nodes to verify.
- **Write File** - A user having write access can change the contents of the file, and to do that, the nodes should first ensure that he does have the write access. The user sends a zero-knowledge proof for the knowledge of e_1 and A. The nodes check this zero-knowledge proof for each accumulator in the array of accumulators corresponding to the file. If the proof is valid for any accumulator, then the user is allowed to change the encrypted file in the blockchain.


Owner adds the decryption key to the TEE
Owner encrypts the file using RSA encryption
Owner send the encrypted file and u, u^e to the blockchain
TEE interacts with the blockchain to get latest accumulator values. If the reader has access it decrypts the file
Owner adds private key of the user to the accumulator in blockchain
Writer send the zero knowledge proof of its access. After verification blockchain allows user to change file contents

Future Work

- **Further Security Analysis of Zerocoin** - As mentioned, we need strong mathematical proof to show that the CL Accumulator proof of knowledge is indeed zero-knowledge even when the adversary knows all the valid witness value pair.
- **Improving Efficiency of CL Accumulator-based Access Control** - The Access Control system requires linear space and time to verify if the user has write access. This requires further investigation to improve efficiency. Also, the implementation of TEE and integrating it along with Blockchain requires further study. One method to improve efficiency is to explore proxy re-encryption i.e. to redeliver keys to the reader instead of using TEE. Also the revocation of access also requires the study of proof of no knowledge, and some research in this area.
- **Analysis of zk-SNARKs for Malleability attacks** - Further research should be conducted to analyse the susceptibility of zk-SNARKs to malleability attacks in the context of the Zcash system. Understanding potential vulnerabilities introduced by such attacks will enable us to enhance the overall security of the system and identify any necessary countermeasures.