# Final report

Yu SU    55323707

**Practical application of smart contract in blockchain**

## 1. Introduction

1.1 Background

With the development of bitcoin[12] and erthereum[8] the smart contracts0 in blockchain[14] are more and more popular as a distributed and reliable ledger which allows distrustful parties to transact safely without trusting third parties.

Bitcoin is good, but it can only handle the micro transaction because of its low scalability. What’s more, there are a lot of privacy problem.

While hiding information is very easy using encryption. the hard part is proving and maintaining integrity under the veil of encryption.

A good computation integrity and privacy need to have three features.

Universality: You would like computation and integrity and privacy to cover any program that you want to be able to apply proofs. That program operated correctly as opposed to the case we're only for specific problems.

Scalability: As the running time and complexity of the program grows, the running time of the prover does not explode. So it goes up roughly or closely an almost linear Fashion.

Succinctness: you would like the proofs to be very short and very quick to verify. And this is for two reasons. One is that you may want to use it to compress long computations. Another reason is that you would like the heavyweight process of proving a statement to fall mostly on the prover and not on the parties verifying.

To handle these issues, Eli proposed ZK-snarks[7],and before this, a spectacular body of knowledge that started being discovered in the late 1980s and led to many awards including curing awards. This is one of the most spectacular achievements of computer science in the past half-century, zero knowledge proof (section 2.2) which allow you to effectively solve the problem in a cryptographic way and extremely efficiently.

And finally Eli used ZK-snarks in Z cash[5] achieve all of the above and they're really efficient, which are based on some cutting-edge cryptography called quadratic arithmetic programs and bilinear pairings.

But a new problem come out. In order to use these ZK-snarks a trusted set up phase is needed in which a select party is or a select set. Trusted set up phase maybe looks harmless. However, with increased value there's going to be increased incentive. There is gonna be a huge huge incentive to update like generate parameters again, when the fortune becomes large. Even if the updater is completely fair, other trust problem which is related to the citizen right and liabilities will restrict its scalability to other field. So there need to be a new approach to solve this problem. Luckily, transparent can solve it perfectly.

Besides solving the privacy problem, we also need to solve the problem of scalability.

The hardest part in this process is to solve the NP problem. The NP problem is the problem is hard to get the result but you can verify the answer easily. Exactly, this is what we want use in the blockchain.

Therefore, to decrease the complexity of time in the proof. Eli constantly optimizing for proof, such as PCP(section 2.1) and IOP(section 2.3). And in the ZK-stark[4], a new biased RS-IOPP(section 3.2.1) was created.

## 1.2 motivation

Eli Ben-Sasson proposed a ZK-STARK structure for the DNA profile match problem in his paper 'Scalable, transparent, and post-quantum secure computational integrity' in March 6,2018.

ZK-STARK is the best approach for guaranteeing the privacy in the transaction of the smart contract up to now.

The privacy is necessary in the application of smart contract in blockchain. Because the entire sequence of operations taken in the smart contract is spread over the network and recorded on the blockchain, it is publicly visible. Although the parties can create new pseudonymous public keys to increase their anonymity, the values of all transactions and balances for each (pseudonymous) public key are publicly visible[11]. Therefore, we need to find an approach to solve this problem. Because it is not only about privacy, but also the security and other scalability problem.

Thanks to Eli Ben-Sasson's study, a new more effective and secure method to keep the privacy in smart contract is proposed. However, there is still no any implement applying ZK-STARK in practice. So It makes a lot of sense to do some research on it. In this paper, I studied ZK-STARK and did some summary. This research is meaningful to find out the drawbacks of the ZK-STARK and also for implementing ZK-STARK in practice in the future.

## 1.3 object

In this paper, there are three main objects.

First is I studied the ZK-STARK and understand the math and logical theory of it.

Second is I do some summary of ZK-STARK like making compare with other approach, showing its merit and demerit and a prospect forecast to how to make it in implement.

Third is design a simple architecture for implementation on ZK-STARK

## 2. Related work

In this section, what I mainly introduce is the foundation about ZK-STARK and the development of the proof system.

### 2.1 Probabilistic checkable proof[2]

First of all, I want to introduce the the probabilistic checkable proof and probabilistic checkable proof system.

For the NP problem, it is not easy to get the answer directly. Therefore, we want to generate one proof model to prove if the statement is true. Before we import the probability, we need to initialize a fundamental proof system.

First we define a decision problem *L*(or a language L with its alphabet set Σ).

Then, we get a probabilistically checkable proof system for *L* consists of a prover and a verifier. After it, we input a claimed solution x with length n.

We judge it is true or false. If it is true($x \in L$, that means the proof is a string $\in \Sigma^*$), the prover produces a proof $\pi$ which states *x* solves L. Then verifier accept the statement, which is called completeness. If it is false, the verifier just does not accept it.

This is the original proof system. But it is really unscalable. We found if the system only solves the problem in a specific probability. The result is still always right and the performance of the system is excellent.

So we import the probability into the two property.

One is the completeness marked as $c(n)$ which is the probability that the verifier accepts the statement at least with.

The other one is soundness marked as $s(n)$ which is the probability that the verifier mistakenly accepts the statement at most with. and $0 \le s(n) \le c(n) \le 1$.

To calculate the computational complexity of the verifier, we import two complexity.

One is *randomness complexity* $r(n)$, which is the maximum number of random bits that *V* uses over all *x* of length *n*.

And the other one is *query complexity* $q(n)$, which is imported to measure the maximum number of queries that *V* makes to π over all *x* of length *n*.

The proof of running time could be made as close as possible to T which is the running time of the original problem program. So if you want to run in time T you're going to have query complexity your proof length that scales like a log of T which is good. But to some power that goes up as you make the running so the shorter the prover running time the bigger the query complexity. To make it more scalable The complexity. We defined[9]:

Completeness x ∈ L ->Pr[V(x)<->P(x)->accept] =1

Soundness x ∉ L -> Pr[V(x)<->P(x)->accept] <=1/2

The complexity class PCP is defined as PCP(1,1/2) [O(log *n*), O(1)]. It is very impressive.

## 2.2 Zero Knowledge proofs[7]

To guarantee privacy what we need is zero-knowledge proof.

Zero knowledge proofs were first conceived in 1985 by Shafi Goldwasser, Silvio Micali, and Charles Rackoff in their paper "The Knowledge Complexity of Interactive Proof-Systems"

The different between the zero knowledge proof and other original proof system that in the process of proving, the prover does not need to convey any information about the process but it also can make the verifier believe the statement is true, which is the key of keeping privacy in the zero knowledge proof.

Like most proof system, it has Completeness and soundness, which are no more different.

But it has an unique character called zero-knowledge. That means no one knows anything, but the verifier could give the judgment if the statement is true by given some simulators, which can produce a transcript that is similar to the interaction between the verifier in question and the honest prover.

A formal definition[13] of zero-knowledge:

View $V[P(x) \leftrightarrow V(x)]$ is the record of a original interactive proof, input is x and the there is a proof<P,V> for language L.

Because it uses zero knowledge for any PPT verifier V∗ there exists an expected PPT Simulator S for the language L

$\forall x \in L, z \in \{0,1\}^*, \text{View } V^*[P(x) \leftrightarrow V^*(x, z)] = S(x, z)$

This formula shows the reproducing process in zero knowledge proof.

z is the prior knowledge, which is the auxiliary argument for V*; V* and P can reproduce the conversation only if S and V* has the same z.

And the other part of the formula is similar to the original interactive proof. This structure of proof make it is possible to realize conversation do not need to convey any information but z. in the zero knowledge proof, only if the verifier V* exists an efficient simulator S, the interactive proof system can reproduce the conversation between P and V* whatever any input, which satisfies the requirement of the privacy.

While after we import ZK things are going to become much worse. Even if you have it at the best of cases you would have even for your Millia for this million lying program you would have a proof whose length is much more than this is like four terabytes and this is under very optimistic assumptions it would have been much worse. so it wasn't really scalable.

## 2.3 IOP[6]:

To make the proof more scalable , I introduce a proof called interactive oracle proof. interactive oracle proofs (IOPs).This model naturally combines aspects of IPs[3] and PCPs, and also generalizes IPCPs[10]; namely, an IOP is a “multi-round PCP” that generalizes an interactive proof as follows: the verifier has oracle access to the prover’s messages, and may probabilistically query them (rather than having to read them in full).

We show that IOPs characterize NEXP (like PCPs); both sequential and parallel repetition of IOPs yield (perfect) exponential soundness error reduction(like IPs); and

any IOP can be converted into a public-coin one (like IPs). These basic complexity-theoretic properties confirm that our definition of IOP is a natural way to combine aspects of PCPs and IPs, and to generalize IPCPs.
An IOPS for a relation R with round complexity k:{0,1}∗→N and soundness s:{0,1}∗→[0,1] is a tuple(P,V), where P,V are probabilistic algorithms, that satisfies the following properties.

The completeness is different for every(x, w)∈R, 〈P(x, w),V(x)〉 is a k(x)-round interactive oracle protocol with accepting the probability.
The soundness is different for every x ∉ L(R) and P,〈P,V(x)〉is a k(x)-round interactive oracle protocol with accepting probability at most s(x).
The more detail about the IOP will discuss in the section 3.2.1 with the FRS-IOPP.

## 2.4 Merkle tree[21]

Merkle tree is also called hash tree, because it is a kind of tree data structure, it is proposed in 1979. In hash tree, each label of the leaf node is the hash of the data block and the label of the non-leaf node is the hash of the hash of their children. It is very efficient and secure for the big data, so it is applied in the blockchain widely.
Hash tree is a kind of one orientation hash. If you want to verify if one data is in the hash tree, you don't have to know all the leaf nodes to figure out the root hash. Because every parent hash must be hashed by two children, so we just need to pick out all the nodes that participate to prove that this leaf node exists in the tree. This reduces the number of hash operations. And the nodes that are selected, and the hierarchy between them, are the verification paths.
In our paper, we use Merkle tree to store the input data, which is efficient to retrieve or verify a specific value from lots of the data.

# 3. Basic math knowledge

## 3.1 Lagrange Interpolation Polynomial[22]

In numerical analysis, Lagrange interpolation is a polynomial interpolation method named after Joseph Lagrange, a French mathematician in the 18th century. In many practical problems, functions are used to express some internal relations or laws between the results, while many functions can only be understood through complicated experiments and multiple observations. However, if a physical quantity in practice is observed and corresponding observed values are obtained in several different places, the Lagrange interpolation method can find a polynomial, which happens to get the observed values at each observed point. Polynomials such as the one above are called Lagrange polynomials.

For a polynomial function, it is known that there are given k + 1 value points:

(x0,y0)…(xk,yk)

Where xj corresponds to the position of the independent variable, and yj corresponds to the value of the function at this position.

Assuming that any two different xj are different from each other, the Lagrange interpolation polynomial obtained by applying the Lagrange interpolation formula is

L(x) = y0l0(x)+y1l1(x)+…+yklk(x)

Lj(x) = (x-x0)/(xj-x0)…(x-xj-1)/(xj-xj-1)

So at each xi, L(xi) = yi +0 +0 +…0= yi

## 3.2 Fast Fourier transform[23]

Finite length sequences can be discretized into finite length sequences by means of discrete Fourier transform (DFT). In 1965, Cooley and Tukey proposed a fast algorithm for calculating the discrete Fourier transform (DFT), which reduced the computation amount of DFT a lot. Since then, the research on fast Fourier transform (FFT) algorithm has been deepening, and digital signal processing, a new subject, has been developing rapidly with the emergence and development of FFT. According to the difference of sequence decomposition and selection method, FFT algorithms are generated. The basic algorithms are base 2DIT and base 2DIF.FFT also has important applications in inverse discrete Fourier transform, linear convolution and linear correlation.

## 3.3 Extended Euclidean algorithm[24]

Extended Euclidean algorithm is an extension of Euclidean algorithm (also known as division by tossing). Given the integers a and b, the extended Euclidean algorithm can find the integers x and y (one of which is likely to be negative) at the same time of finding the maximum common divisor of a and b, so that they satisfy the Bayesian equation:

A, x plus b, y is equal to GCD of a, b.

If a is a negative number, we can turn the problem into |, a | (minus x) + b y = GCD (|, a |, b) (|, a | is the absolute value of a), and then set x '= (minus x).

Usually when we talk about the greatest common divisor, we will mention a very basic fact: given two integers a and b, there must be integers x and y so that ax + by = GCD (a , b).

There are two Numbers a and b, and if you divide them by tossing and turning, you get their greatest common divisor -- that's what we know. And then, if you take the expression that's generated by the flip division, and you go back, you get an integer solution to ax plus by is equal to GCD of a,b.

Extended Euclidean algorithms can be used to compute mode-inverse elements (also known as mode-inverse elements), which play an important role in RSA encryption algorithms.

## 3.4 Fermat's little theorem[25]

Fermat’s little theorem is an important theorem in number theory, put forward in 1636, its content is: if p is a prime, and the GCD (a, p) = 1, then a (p - 1) ≡ 1 (mod p), for example, if a is an integer, p is a prime, then a, p obviously co-prime (i.e., there is only one common divisor both 1), so we can get a special case of Fermat's little theorem, namely when p is prime time, a ^ (p - 1) ≡ 1 (mod p).

Because the modular model is a finite field. So from the Fermat's little theorem we could know that the If p-1 is a multiple of some number k, then in the circulation, the result of x^k will show the duplicate. So the number of the passible result of the function is (p-1)/k +1.

# 4. ZK-STARK structure

## 4.1 Introduction

ZK-STARK(Zero-knowledge Scalable Transparent ARguments of Knowledge) was proposed in 2018 by Eli Ben-Sasson to solve the most of drawbacks of ZK-SNARK, the general-purpose succinct zero knowledge proof technology that can be used for all sorts of use cases ranging forms verifiable computation to privacy-preserving cryptocurrency.

There are two main advantages of ZK-STARK:

First is the trusted setup phase. n ZK-STARKs, there is no external trusted setup phase and the randomness used is public information.

The other one is about Scalability. ZK-STARK has lower complexity in following four aspects.

1. Arithmetic circuit complexity
2. Communication complexity
3. Prover complexity
4. Verifier complexity

## 4.2 Method

In this section I want to introduce zk-stark's main innovative components, which support double scalability and specific efficiency.

At first I want to introduce the polynomial proof, this is the basic of the whole system. Then, I will introduce the Fast Reed-Solomon Interactive Oracle Proof of Proximity (FRS IOPP)[4](section 3.2.1), which is a new proof using in ZK-stark. it is very scalable. And I also will introduce two kind of arithmetization, AIR and ALI to optimize the complexity of the prover. After that, I will introduce the LDE, which reduce the cost of the verifier. At last, I will introduce the approach which could reduce the APC and CC. after all this, I will focus on the zero knowledge that achieved in our system.

1. Proofs with Polynomials[26]

In the traditional proof method, if we want to verify something we need to put all the data to verify. However, when the dataset is very large, the computation cost becomes very high. For example, if there is a blockchain like Ethereum, and you want to verify if one of the blocks and other block in its blockchain is valid. You need to input each bit of these block. It is unreal to realize. So we want to find some way only to verify the part of the data, but can still verify the proof well.

Besides the size, another problem is to provide succinctness. It is hard to find a way to find a one-bit error in a very long data. What we want is a succinct way which is very sensitive to the error.
So we import the proof with polynomials, which is the premise to realize all the demand. and converted our original statement into polynomial which looks mathematically clean and possibly quite provable.

2. constraint checking polynomial
After using the polynomial, we want to regulate a method to verify the polynomial. First we define an function P(x), we w ant to verify for all the x from a to b, P(x) is in the range K.
To verify it, we define an verification function, C(x), for every x belong in range K, the C(x) equal to 0. This is easy to realize in polynomial, just like C(x) = (x0-x) * (x1-x) * … * (xk-x).
To make calculate easier, we import Z(x), Z(x) is similar to C(x) , while the range in xi is associated with x but not P(x). So any polynomial in this range is a multiple of Z(x). And we import another verification argument D(x).
As the relation between the C(P(x)) and Z(x), we could get C(P(x)) =Z(x) * D(x).
That means, in the verification phase, we only need to calculate the input whether it satisfies this formula.

However, the computation cost is extremely high, and when the polynomial becomes very long, it is hard to be succinct. So we want to find a kind of finite field which could makes data has an bound but not increase infinitely. Specially in this system, we use the modular arithmetic as the possible finite field. In this way, the number of the data will not be out of the prime N.

And because we need to do some division operation in the proving phase, we import the modular inverses to reduce the computation. To calculate the modular inverses, the best algorithm is Extended Euclidean algorithm (which I have mentioned in the section 3).

After all the optimization, our original model is ready to be verified in the way of polynomial succinctly. But we still do not solve the cost of the computation problem. So we want to use the proximity test to reduce the computation.

## 3. basic process

After we describe the proof of polynomial, we will introduce the specific process of STARK proof.

First, in the preparation phase, we set a prover P and a verifier V, where we set a function C(x), just like we mentioned in the part 2.

And we assume that there is a public key Z(x) which I have mentioned in the constraint checking polynomial part, we could use the Z(x) to verify if the data provided from prover is the the same as the one in the proof. Although everyone could get the Z(x), it is not stored in the verifier. The part is stored is the Merkle root of Z(x), when doing some query, the verifier will ask prover commits the branches of Z(x) that needed.

Then the prover commits a Merkle root of P(x) and D(x) for all the x to verifier and wait for the response.

After receiving the Merkle tree root, it asks the prover commits some branches of Merkle tree in random.

And after the prover commit the data, the verifier check if the branches is associated with that Merkle tree root, if yes, it verify if it satisfies the formula “C(P(x)) = Z(x) *

D(x)"(the reason I have mentioned in part 2).

*Completeness:* if you actually know a suitable P(x), then if you calculate D(x) and construct the proof correctly it will always pass all 16 checks.

*soundness:* Because C(P(x)) is a degree-10 polynomial composed with a degree-1,000,000 polynomial, its degree will be at most 10,000,000. In general, we know that two different degree-N polynomials agree on at most N points; hence, a degree-10,000,000 polynomial which is not equal to any polynomial which always equals Z(x) * D(x) for some x will necessarily disagree with them all at at least 990,000,000 points. Hence, the probability that a bad P(x) will get caught in even one round is already 99%; with 16 checks, the probability of getting caught goes up to 1 - $10^{-32}$; that is to say, the scheme is about as hard to spoof as it is to compute a hash collision.

However, this way could only magnify the error and do the verification under one premise that this point is the word to be verified is actually on the polynomial of C(x). Because any C(x) and Z(x) could get a d(x) satisfying the condition. And send the C(x) and d(x) to verifier, it will pass the verifier.

So besides the basic proof process, what we need to do more is finish the *proof of proximity, which could confirm that most of the test point belongs to the correspond polynomial.*

4. proof of proximity[27]

To confirm the data we input is correspond to the related polynomial, we import proof of proximity to confirm there must at least some of the points that satisfy the probability are on the same polynomial.

However, the problem is the if we want to verify a polynomial whose degree is less than D, we at least need D points to verify. Because D point could just define an unique polynomial whose degree is less D, so we need use more than D points to verify.

So here we import the Lagrange interpolation (I have mentioned in section 3). We choose a subset of D, using the Lagrange interpolation to recover the polynomial.

And check against the other points if they are in the same polynomial.

But the problem is if the D is very large, this operation becomes very unscalable. Luckily, STARK provide an approach called Fast RS IOPP, which could solve this problem quickly and efficiently.

5. Fast Reed-Solomon Interactive Oracle Proof of Proximity[28]

To understand what is FRI, first we need to know the Reed-Solomon codes.

RS code is a group of error-correcting codes, which could detect and correct multiple symbol errors and it is appropriate for the finite field model just like the modular model we used.

In detail, If there is *t* check symbols in the data, RS codes can detect at most *t* erroneous symbols and also their any combination, or correct at most *t*/2 erroneous symbols and also their any combination.

Then we need to look back at the IOP that I have mentioned in the related work section.

An Interactive Oracle Proof (IOP) system S is like the other interactive proof system. First, we define a prover P and a verifier V for this system. So S could be denoted as S = (P, V). The difference between the original IP system is that there are many round in IOP. So we define r(n) is the round complexity of the system, n is the length of the input x. At every round of interaction, the verifier sends over a message. The prover also replies to the message, except that now the verifier has the freedom to probabilistically sample it rather than read it in its entirety.

Then we could introduce the fast RS IOPP Theorem 2(Main – FRI properties).The binary additive RS code family of rateρ= 2−R, R ≥2, R ∈ Nhas an IOPP (FRI) with the following properties, whereN=|S| denotes block-length (which equals the prover-side input length for a fixedRS[F, S, ρ] code)and ρN > 16:

Prover Complexity is less than 6Narithmetic operations inF; proof length is less than N/3 field elements and round complexity is at most log N2; In this new model, we can solve the low-degree testing problem and make the arithmetic complexity is strictly linear.

That means we can prove proximity to degree < D polynomial with less than D queries.
However, if we want to realize it, besides this model, we still need a arithmetization approach, called Algebraic Intermediate Representation.

## 6. Algebraic Intermediate Representation

Constructing an algebraic intermediate representation (AIR) is the first phase of arithmetization, which is similar to the program compile processes.
AIR is one kind of intermediate representation to make the compile process more efficient. In detail, we could decompose the polynomial f(x) to a bivariate polynomial g(x,y).
There are two reasons: first one is it has efficient arithmetic operations. And the other reason is its algebraic structure is needed for the FRI3rd protocol.
As a result, in each one round verification:
the data prover commit is not the Merkle tree of f(x) anymore, it becomes the Merkle tree of the g(x,y).
And after receiving the commit, the verifier randomly picks a few dozen rows and columns and each row and column must contain some number of the point(at least one point is on the diagonal)
The prover do not reply p(x) either, instead, prover send all the Merkle branches that about all the points.
The verifier verify the g(x,y) in the way of STARK to prove all the points that prover provides correspond to the related degree polynomial.

This gives the verifier a statistical proof because most rows and columns and most part of the diagonal are populated mostly by points on degree < 1000 polynomials.

This thus convinces the verifier that most points on the diagonal actually do correspond to a degree < 1,000,000 polynomial.

Apparently, the number of r rows * c columns * p points is smaller than the original degree but not by that much, the complexity of verification is still sublinear. What's more, because the AIR phase, the complexity of prover becomes larger. So we still cannot say this is scalable enough. We still need to make some optimization about this proof. So we import the Algebraic Linking Interactive Oracle Proof to reduce the prover complexity.

## 7. Algebraic Linking Interactive Oracle Proof[29]

The main cost for prover is because applying AIR the prover need apply a local map to provide the bivariate polynomial. Thanks to the Fermat's little theorem(mentioned in the section 3), we know in the local map many parts is duplicate. So we could remove the duplicate part using ALI.
Algebraic linking interactive oracle proof is a one round iop, which reduce the computation of the y from the g (x,y). it only uses a single round of interaction instead of the all s rounds of constraint. Beside the prover time, we also want to reduce the verifier complexity, so we import Low degree extension and composition degree.

As it turns out, we can go further: we can have the prover only commit to the evaluation of g on a single column. The key trick is that the original data itself already contains 1000 points that are on any given row, so we can simply sample those, derive the degree < 1000 polynomial that they are on, and then check that the corresponding point on the column is on the same polynomial. We then check that the column itself is a < 1000 polynomial.

The verifier complexity is still sublinear, but the prover complexity has now decreased to $10^9$, making it linear in the number of queries (though it's still superlinear in practice because of polynomial evaluation overhead).

## 8. Low degree extension and composition degree[15]

The main cost of verifier is inducing by the polynomial interpolation. The cost of it increase with the increase of the input size.

Because the data is a kind of finite field, we could use additive fast Fourier transform in the multiple of polynomial. We could reduce the degree of the polynomial by many times recursion. Each step, we make the degree of the process of proving proximity reduced by half. So the prover's computational complexity goes down by a factor of 2 each time

But there are some different in zk-stark from the FFT, we only introduce one new sub problem but not both of two.

Until the column is small enough to be calculate, we could easily verify the problem.

## 9. Minimizing Authentication Path Complexity (APC) and Communication Complexity (CC)

Besides the prover and verifier complexity, we also want to get some optimization from the authentication path complexity and communication complexity.

1. we want to reduce the authentication path in the verifier querying process. As we have known, in the LDE section, we will use one row of original data as the input of the Lagrange interpolation. So when we verify the p(x) if it is from the given Merkle root, it is more efficient if all point related to one row is under the same father node. So we make prover place one row in a sub-tree, which could reduce the complexity of verifier.

2. from the first optimization, we get an idea. We could merge the same set in one same Merkle sub-tree which could reduce the complexity of the communication. When the verifier query the data from the prover. FRI protocol always affine cosets of a fixed subspace, so we can put on coset in one sub-tree.

After all this, we choose the Davies-Meyer hash function composed with the AES as our hash function, because is could improve the running time and also reduce the complexity of communication.

10. zero knowledge[30]

In zk-stark, we realize the perfect zero knowledge, which is not a statistical zero knowledge but the computational zero knowledge.

So let me first define by picture what I mean by perfect zero knowledge interactive PCPs and interactive Oracle proofs.

Now in principle, it could have two edges over the prover. It could rewind the verifier, or it can also adaptable answer queries.

First and foremost, from a technical perspective, protocols that are perfectly secure with straight line simulation satisfy general concurrent composition. This is not true for statistical zero knowledge, and much less computational zero knowledge.

Second, from an efficiency perspective, perfect zero knowledge doesn't
have this security parameter that governs how far away the two views are from each other.

It will rely on techniques from algebraic complexity theory. In fact, the randomization techniques, which sounds really odd.

So we will run into a polynomial Q. It's going to be related to F. And I'm going to hide Q behind that grey curtain to mean that Q is going to be like a virtual oracle that is induced by actual oracles.

Now the actual prover and the actual verifier literally do have access to F. And now we just need to decide all the prover does.

He samples random a multilinear polynomial that adds up to 0 on the Boolean hypercube and sends it as his oracle.

The verifier samples a random field element from F and sends it over to the prover. Q is defined as a mask of F. So you take F, you multiply it by delta, and you add R.

Q is not an actual oracle. The verifier can simulate Q on any point by querying R and F.

In the application of sharp P, the oracle F is succinctly represented by an arithmetized Boolean formula,

Completeness does make sense. If the sum of F is 0, the sum of R is 0, then the sum of Q if 0.

For soundness, let's assume for a moment R is multilinear for real. Then there's at most one exists actually unique delta that makes the sum of Q equal 0. On all other deltas, the sum of Q will be nonzero, and the prover verify will interact on a false statement.

So via union bound, the verifier will accept with small probability.

Look at Q. We took F, multiplied it by something, and added R. R is random, I mean completely random. So Q carries no information about F.

Now this is strong intuition, but it's actually technically very far from being true.

First and foremost, Q it's not random.

The verifier is opposed to p ick delta at random, but he can query R at a number of points and picked delta depending on R. That would make Q not random.

Secondly, even if the malicious verifier did pick delta independently from R and sent the honest delta, now the simulator potentially has a daunting task. In polynomial time, he has to make up answers about the multilinear polynomial, whose table is enormous. It has F to the m values And just sampling a multilinear
polynomial takes F to the m.

More than that, it has to be able to conduct and fake answers about a sumcheck, about this polynomial Q that depends on R and F as well.

So it turns out, actually, this exact protocol is perfect zero knowledge.

## 5. Compare

Here we want to talk about the difference between different realized proof system to emphasis the advantage of zk-stark. The most of the data is import from the original paper of zk-stark[4].

• Discrete logarithm problem (DLP)[20]:

A method proposed by Groth , which is implemented in, relies on the hardness of DLP to build a transparent system. Shor's quantum decomposition algorithm effectively solves the problem of DLP and makes the method have quantum susceptibility. In addition, validator complexity in DLP methods requires time greater than TChence is for (according to our definition of the term), although in DLP communication complexity methods are logarithmic. We call the initial implementation of the system BCCGP and the most recent improved version bulletproof.

• Interactive Proofs (IP) based(mentioned in section 2):

IP protocols can be implemented with zero knowledge, but only recently have IP protocols been effectively "reduced" to small (non-sequential) depths of computation through what Goldwasser et al have called "muggle proof" . This led to a series of implementations of the code. Earlier works lacked ZK, but the most advanced works, such as and Hyrax, had ZK.
Like zk-stark, these latest ip-based proofs are transparent and have a scalable demonstrator, but quantum sensitive, and their validators are not scalable because they are "standard"

• Secure multi-party computation (MPC)[18]:

This method is suggested by Ishai et al. and the implementation of the first ZKBoo

system, and more recently, in Ligero, the "compilation" of the security policy committee agreement zk-pcp system requires the verifier to commit to the security policy committee agreement's transcript and then reveal one party's point of view.
Like zk-stark, mpc-based proofs are transparent, post-quantum safe, and have scalable (quasi-linear) proof times. However, a system based on MPC has a non-verifiable verifier, and a system whose running time is greater than TCand communication complexity is not extensible. It is a system whose art state is square root of TCin. However, it is very effective for specific circuits and amortized calculations.

- Homomorphic public-key cryptography (hPKC)[16]:

This method was proposed by Ishai et al.(for the "designated verifier" case) and Groth[60](for the "public verifiable" case), using an efficient information theory model called "linear PCP", which was then "compiled" into a cryptographic system using hPKC. Gennaro et al. proposed a very effective instantiation method based on quadratic span program. It, for example, prove the system Zerocash and Zcash ™.First implementation based on QSP system, called Pinocchio, and then the implementation of the including libSNARKnfor Zerocash and Zcash ™ implementation.

HPKC lacks the function of the transparency and post-quantum security while the advantage is its verification time in is scalable.

- Incrementally Verifiable Computation (IVC)[17]:

Valiant proposed this method to reduce the space consumption of the proof program by relying on the hypothesis of knowledge extraction. This method can be applied to other proof systems of concise (sublinear) validators, including zk-stark, but only a single hPKC system has been implemented so far.
In contrast to zk-stark, the system built in this way inherits most of the attributes from the underlying proof system. In particular, IVC based on HPKC is

non-transparent and quantum sensitive; However, the validator is scalable even for calculations that are performed only once, because the setup phase runs in log time.

## 6. architecture of implementation

Next, I want to design an architecture to implement the zk-stark.

STARKs ("Scalable Transparent ARgument of Knowledge" are a technique for creating a proof that f(x)=y where f may potentially take a very long time to calculate, but where the proof can be verified very quickly. A STARK is "doubly scalable": for a computation with t steps, it takes roughly O(t * log(t)) steps to produce a proof, which is likely optimal, and it takes ~$O(\log^2(t))$ steps to verify, which for even moderately large values of t is much faster than the original computation. STARKs can also have a privacy-preserving "zero knowledge" property, though the use case we will apply them to here, making verifiable delay functions, does not require this property, so we do not need to worry about it.

Because the time is limited, what I do is only to design the architecture and state the theory concept about each part in this architecture. In other words, I only use pseudocode to introduce each function instead of the real code.

The main architecture is shown like the figure 1 and figure 2.

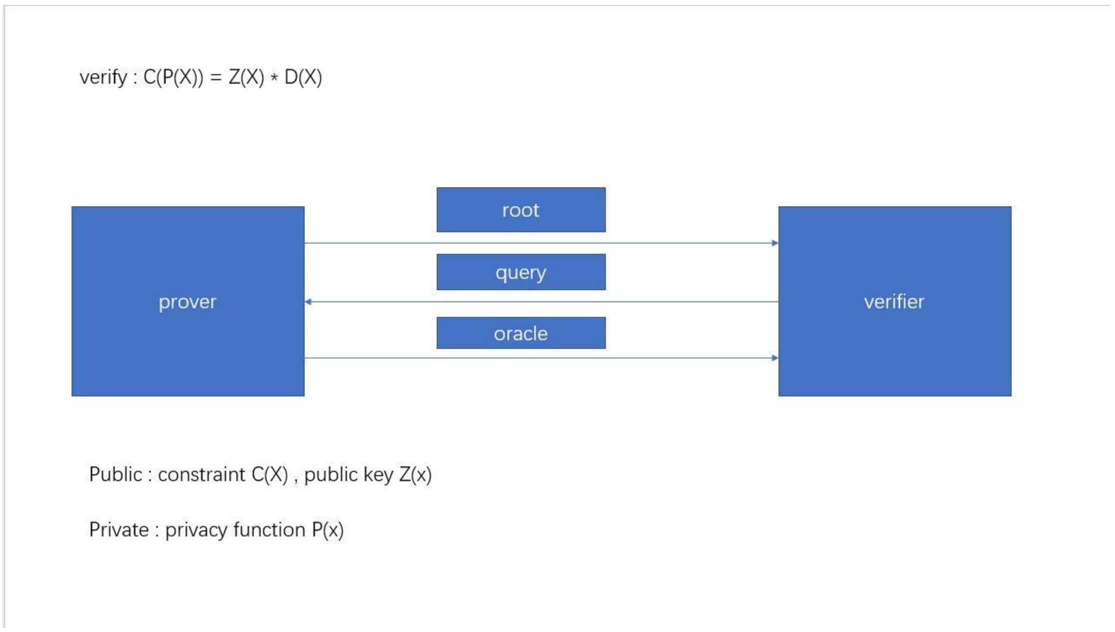
verify : C(P(X)) = Z(X) * D(X)
prover
root
query
oracle
verifier
Public : constraint C(X) , public key Z(x)
Private : privacy function P(x)

figure 1 overall architecture

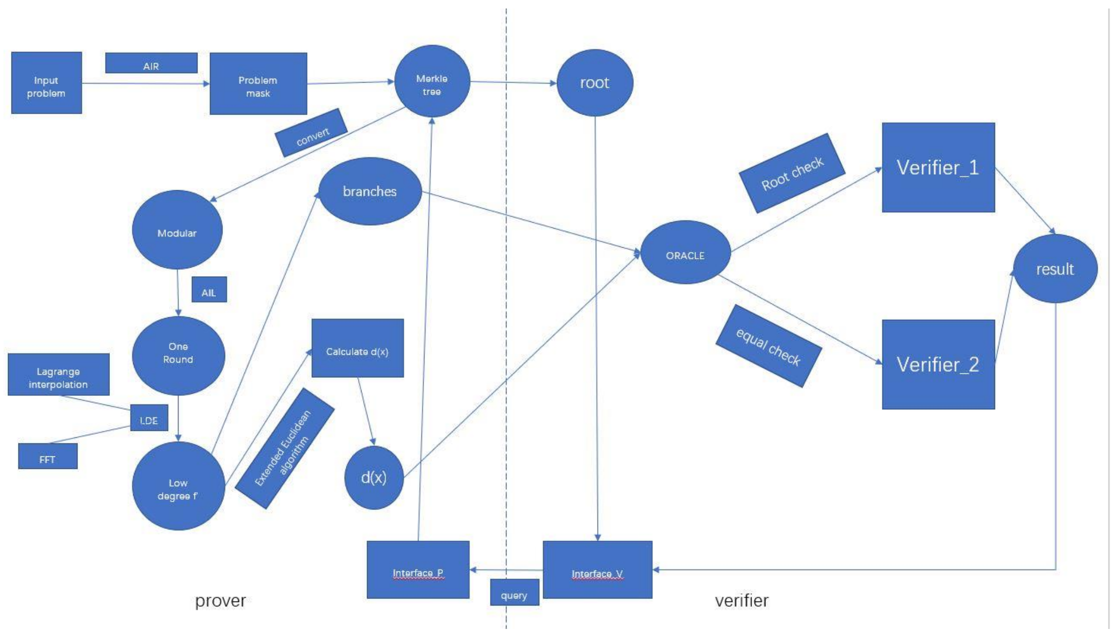

figure 2 detailed architecture

We only focus on the inner part of prover and verifier.

So we simply define input p(x), constraint c(x) and don't think about the content of these function.

6.1 Overall class introduction:

Class modulus_calculation():

This class contains the function of the calculation function for modulus. Like add, sub, multiple, division and also the function of mod.

Class polynomial_calculation():

This class contains the function of the calculation function for polynomial, like add, sub, multiple(there are no division, because use the inverse is more efficient)

Class modulus_optimization():

This class is trying to reduce the cost of the calculation of modulus.

This class contains the extended Euclidean algorithm and Montgomery algorithm to calculate the inverse and multi-inverse respectively.

Class polynomial_optimization():

This class is trying to reduce the cost of the calculation of polynomial.

It contains the function of Lagrange interpolation, Fast Fourier transform and inverse Fast Fourier transform to reduce the complexity of input.

Class prover():

This class contains the most of process in prover.

It contains the function of change the input to bivariate polynomial modulus ,making the Merkle tree and also the function for reducing the complexity of the data.

Class verifier():

Thish class contains the most of process in verifier.

It contains the function of the verifying.

6.2 detailed class introduction

1. class modulus_calculation():

```
    Def polynomial modulus(p,x)
        For each p mul related x get the sum
        Sum modulus
    Def modulus_add(x,y)
        First add, then modulus
    Def modulus_sub(x,y)
```

```
        First sub, then modulus

    Def modulus_mul(x,y)

        First mul, then modulus

    Def modulus_did(x,y)

        Z = Def extended Euclidean algorithm(y)

        Def modulus_mul(x,z)

2.  class polynomial_calculation()

    def add_pol(x,y)

        for each bit add

    def sub_pol(x,y)

        for each bit sub

    def mul_pol(x,y)

        for each bit of x

            for each bit of y

                mul

3.  class modular optimization():

    def extended Euclidean algorithm(x)

        if gcd = 1

        recursion of inverse of Euclidean algorithm
        return inverse

    def Montgomery batch inversion (x)

     partials = [1]
     for i in range(len(values)):
          partials.append(self.mul(partials[-1], values[i] or 1))
     inv = self.inv(partials[-1])
     outputs = [0] * len(values)
     for i in range(len(values), 0, -1):
          outputs[i-1] = self.mul(partials[i-1], inv) if values[i-1] else 0
          inv = self.mul(inv, values[i-1] or 1)
     return outputs
```

4. class polynomial optimization()

```
        def Lagrange interpolation(xs,ys)

            convert each coordinate to factor in the array x

            denominator = Montgomery batch inversion (x)

            for each x += related y * denominator

            return array [x]

            class fast Fourier transformation():

            def FFT(x,modulus,root)

                if len(x) small enough

                    return x
                divide x to 2 part

                for each bit of half part

            for i, (x, y) in enumerate(zip(L, R)):

                y_times_root = y*roots_of_unity[i]

               o[i] = (x+y_times_root) mod

               o[i+len(L)] = (x-y_times_root) mod

            return o

        def inv_FFT(x,modulus,root)

              invlen = f.inv(len(vals))
              return [(x*invlen) % modulus for x in
              fft(vals, modulus, f.inv(root_of_unity))]
```

5. class prover()：

```
    Def Merkel tree()

       randomly select y indices to sample
       create Merkle tree as the hash function of the reduced g(x,y)

    Def ORACLE_AIR_ALI(xs)

        Convert F(x) to g(x,y) and modulus

        Make y[] reduced by modulus

        Call fft make it small
```

6. class verifier():

```
        input -> p(x) is a polynomial

        xs = polynomial modulus(p(x)) // finite field

        y = ORACLE_AIR_ALI(xs)    // convert to linear coordinate

        d = merklize(y) // get the branches

        for each d:

            q = Lagrange interpolation(d)

            d(x) = q * z(x)^-1

            if all x satisfying the q = d(x) * z(x)

            return accept

            else return reject
```

## 7. Conclusion

In this paper, I did a lot of research on the newest proof system, ZK-STARK , including the method and principle of each part. And I also get to know many basic arithmetic knowledge which could improve the scalability of the system.

After that, I did some comparison on the existing proof system to make sure the advantage of the ZK-STARK.

And I also design an efficient architecture with the ZK-STARK system.

My architecture could use in many areas as the proof system, which is scalable, secure, private and transparent.

The problem is totally private for the verifier and the verifying process is completely transparent. But it could still keep secure and efficient.

However, there are still some unsolved problems, like the size of this system is really huge, which makes really implement on it is very difficult. And this problem is not solved until now.

Besides the system, because the time is limited some math knowledge I can not understand very clearly. Some arithmetic for is just like a black box, what I only know is what is could do, but how it could do. And there are also some error in my understanding of the system. I hope I could have a better understand in the future.

## Reference

[1] Anjana, P. S., Kumari, S., Peri, S., Rathor, S., & Somani, A. (2018). An Efficient Framework for Concurrent Execution of Smart Contracts. Retrieved from http://arxiv.org/abs/1809.01326

[2] Arora, S., & Safra, S. (1998). Probabilistic checking of proofs: a new characterization of NP. *Journal of the ACM*, *45*(1), 70–122. https://doi.org/10.1145/273865.273901

[3] Ben-sasson, E. (2017). Concretely e cient Computational Integrity ( CI ) from PCPs PCP e ciency, (June).

[4] Ben-Sasson, E., Bentov, I., … Y. H.-M. (2017). S., & 2017, U. (2018). Scalable, transparent, and post-quantum secure computational integrity. *Eprint.Iacr.Org*, (693423), 1–83. Retrieved from https://eprint.iacr.org/2018/046.pdf

[5] Ben-Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., & Virza, M. (2014). Zerocash: Decentralized anonymous payments from bitcoin. *Proceedings - IEEE Symposium on Security and Privacy*, 459–474. https://doi.org/10.1109/SP.2014.36

[6] Ben-Sasson, E., Chiesa, A., & Spooner, N. (2016). Interactive Oracle Proofs, (April 2015), 1–48. https://doi.org/10.1007/978-3-662-53644-5_2

[7] Bitansky, N., Canetti, R., Chiesa, A., & Tromer, E. (2012). From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference on - ITCS '12*, 326–349. https://doi.org/10.1145/2090236.2090263

[8] Ethereum Homestead Documentation — Ethereum Homestead 0.1 documentation. (n.d.). Retrieved October 21, 2018, from http://www.ethdocs.org/en/latest/

[9] Hirt, M., Eds, A. S., Conference, I., & Hutchison, D. (2016). *Theory of Cryptography* (Vol. 9562). https://doi.org/10.1007/978-3-662-53644-5

[10] Kalai, Y. T., & Raz, R. (2008). Interactive {PCP}. *Icalp (2)*, 536–547.

[11] Kosba, A., Miller, A., Shi, E., Wen, Z., & Papamanthou, C. (2016). Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts. *Proceedings - 2016 IEEE Symposium on Security and Privacy, SP 2016*, 839–858. https://doi.org/10.1109/SP.2016.55

[12] Nakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash SyNakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System. Consulted, 1–9. doi:10.1007/s10838-008-9062-0stem. *Journal for General Philosophy of Science*, *39*(1), 53–67. https://doi.org/10.1007/s10838-008-9062-0

[13] Pass, I. R. (2009). Parrot_MAM II_3.pdf, 1–5.

[14] The Economist. (2015). The great thing of being sure about things. *The Economist*, *417*, 21–24. Retrieved from http://www.economist.com/news/briefing/21677228-technology-behind-bitcoin-lets-people-who-do-not-know-or-trust-each-other-build-dependable

[15] Negre, C. (2003). FINITE FIELD MULTIPLICATION IN LAGRANGE REPRESENTATION USING FAST FOURRIER TRANSFORM, 320–323.

[16] Armknecht, F., Boyd, C., Carr, C., Angela, J., & Reuter, C. A. (n.d.). A Guide to Fully Homomorphic Encryption, 1–35.

[17] Valiant, P. (2007). Incrementally Verifiable Computation or Knowledge Implies Time / Space Efficiency by.

[18] Halevi, S. (n.d.). Round-Optimal Secure Multi-Party Computation, (1617676).

[19] Moshkovitz, L. D., Forbes, S. M., & Moshkovitz, D. (2010). The Low-Degree Testing Assumption, (1), 1–7.

[20] Jens Groth. Efficient zero-knowledge arguments from two-tiered homomorphic commitments. In Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings, pages 431–448, 2011.

[21] *Merkle, R. C. (1988). "A Digital Signature Based on a Conventional Encryption Function". Advances in Cryptology — CRYPTO '87. Lecture Notes in Computer Science.* ***293****. p. 369. doi:10.1007/3-540-48184-2_32. ISBN 978-3-540-18796-7.*

[22] *Meijering, Erik (2002). "A chronology of interpolation: from ancient astronomy to modern signal and image processing" (PDF). Proceedings of the IEEE.* ***90*** *(3): 319–342. doi:10.1109/5.993400.*

[23] *Van Loan, Charles (1992). Computational Frameworks for the Fast Fourier Transform. SIAM.*

[24] *Knuth, Donald. The Art of Computer Programming. Addison-Wesley.* Volume 2, Chapter 4.

[25] *Long, Calvin T. (1972), Elementary Introduction to Number Theory (2nd ed.), Lexington: D. C. Heath and Company, LCCN 77171950*

[26] Hrubeˇ, P. (n.d.). The Proof Complexity of Polynomial Identities.

[27] Vaudenay, S. (n.d.). Proof of Proximity of Knowledge, 1–21.

[28] Ben-Sasson, E., Bentov, I., Horesh, Y., & Riabzev, M. (2018). Fast Reed-Solomon Interactive Oracle Proofs of Proximity, *134*(134). https://doi.org/10.4230/LIPIcs.ICALP.2018.14

[29] Eli Ben-Sasson, Iddo Bentov, Alessandro Chiesa, Ariel Gabizon, Daniel Genkin, Matan Hamilis, Evgenya Pergament, Michael Riabzev, Mark Silberstein, Eran Tromer, and Madars Virza. Computational integrity with a public random string from quasi-linear PCPs. IACR Cryptology ePrint Archive, 2016:646, 2016.

[30] Eli, E., Michael, E., Michael, F., & Nick, E. (2016). Probabilistic Checking in Perfect Zero Knowledge, 1–47. https://doi.org/10.1016/j.jmmm.2008.