THE JBBA

PEER REVIEWED RESEARCH

OPEN ACCESS
ISSN Online: 2516-3957
ISSN Print: 2516-3949
https://doi.org/10.31585/jbba-4-1-(9)2021

Investment Compliance in Hedge Funds using Zero Knowledge Proofs

Komal Kalra, Shubham Sahai, Sandeep Kumar Shukla
Department of Computer Science & Engineering, Indian Institute of Technology (IIT) Kanpur, India

Correspondence: komalklr@gmail.com
Received: 15 January 2021 **Accepted:** 24 March 2021 **Published:** 5 April 2021

Abstract

Financial Regulation is a form of compliance system that subjects financial institutions to certain requirements and restrictions. Investment Compliance is an example that involves investment restrictions and monitoring on behalf of investors. Hedge Funds differ from other traditional funds such as mutual funds because of their ability to employ complex investment and hedging techniques. These are private entities with few public disclosure requirements. This is useful in a way as the strategies used are confidential which allows financial agents to participate in the financial markets without any fear of information leakage, thereby promoting liquidity. However, this is often implied as the lack of transparency. Hedge Funds are expected to produce higher returns, but sometimes investors seek a risk guarantee in addition to higher returns. However, too much transparency rules out the incentives financial entities have by participating in the first place. On the other hand, too much secrecy may give rise to malicious entities that can break the rules due to a lack of compliance. We aim to solve this problem of protecting investors while ensuring the privacy of financial bodies using zero knowledge proofs. Proofs can be visualised as a way of providing enough information to investors while the zero-knowledge property of proofs maintains the privacy of the fund manager's strategies. We propose a protocol to address this scenario using Zokrates, a framework for verifiable computation using Zk-SNARKs on Ethereum, to encode the constraints and export the verifier. Based on our implementation and analysis, it can be concluded that zero knowledge proofs provide us with a variety of ways to develop compliance systems.

Keywords: *compliance, investors, fund manager, proofs, transparency*

JEL Classifications: *G11, G18, G28*

1. Introduction to Investment Compliance

In the financial context, the term hedge refers to placing limits on risk. The ability to employ complex trading strategies distinguishes hedge funds from other funds. Generally, these are considered risky investments, which is why only accredited investors, investors with high financial sophistication, can make investments in them. Although hedge funds are not subject to many restrictions that apply to regulated funds, guidelines were passed in some countries following the financial crisis of 2008 to increase government regulation of hedge funds. In addition, SEC and other regulatory bodies have requested more transparent hedge fund practices over the years [34, 38].

Hedge Funds are privately owned funds that face relatively fewer regulations and conditions than other funds (e.g. mutual funds and equity funds). To protect investors, there are strict guidelines from regulatory bodies, such as SEC. Few examples would be that only investors with income more than a particular value are allowed, only investors with a net worth exceeding a particular value are allowed, etc. However, investors would also like to ensure that fund managers are behaving properly and that their investments do not exceed the level of risk. On the other end, the fund manager might not want to disclose all their portfolio characteristics as this may lead to leakage of the strategies used by them. Portfolio characteristics for a particular fund describe the allocation of investments in different assets.

We begin by defining zero knowledge proof systems [36], a scheme in which the prover convinces the verifier about the fact that they have knowledge about a particular statement without revealing anything about the statement. Section 2 describes the zero knowledge proofs in detail. Due to the confidential nature of the portfolio and the need to regulate the investment process to protect interest of investors, this problem can be reduced to zero-knowledge proofs. Proofs can be visualised as a way of providing enough information to investors while the zero-knowledge property of proofs helps to maintain the privacy of the fund manager's strategies.

1.2 Related Works

To solve the problem of conflict of interest between investors and fund managers, Szydlo [31], in 2005, described a protocol between investors and fund managers. Precisely, he described the portfolio characteristics and risk factors for each asset and defined a linear condition that is to be proven by the fund manager to convince investors that their risk measure does not exceed any predefined risk threshold. For this, he used Pederson Commitments [36] and Interval Proofs using Shoup's NTL package [37]. Another related work is given by Gowravaram [18] which uses the same method of commitments and Interval Proofs.

1.3 Our Contribution

As there is a lack of trust between the fund manager and investors, there needs to be a way to solve this problem of conflict of interest between parties. Here comes the role of blockchain smart contracts to verify that the fund manager follows the rules specified by the investor (or predefined by the fund manager) without depending upon any central authority. We use Ethereum smart contracts as a form of agreement between two parties such that investors can verify that funds follow the specified guidelines and are behaving properly. For this, we use a zero-knowledge proof systems framework Zokrates (SNARKS for Ethereum), which uses libsnark by Pinocchio protocol (or bellman for Groth16). Libsnark is a C++ library for SNARK systems and provides mechanisms to encode most of the problems in the form of Rank-1 Constraint Systems(R1CS) and then into Quadratic Arithmetic Programs (QAP), from which proofs are generated such that bilinear maps can be used for verification which makes it efficient to verify. To summarise,

- Zokrates framework provides us with the ability to generate the Solidity Contract which can be deployed directly on Ethereum and verification can be performed by calling a method on the contract.
- One can specify any condition (that can be encoded in libsnark) and encode it into constraints so that verification can be performed in constant time and with constant proof size.
- Using this method to encode the constraints also gives us an added advantage to encode quadratic (and higher-degree) constraints that might be required from the financial point of view.

We begin with the definition of zero knowledge proofs and cryptographic preliminaries required for the protocol in Section 2. Section 3 describes Pinocchio Protocol and Zokrates architecture. In Section 4, the problem statement is explained in detail. Section 5 describes the protocol workflow and implementation details using Zokrates. Section 6 presents the evaluation results of the proposed protocol. Finally, in Section 7, we conclude this article and suggest some scope of future work for this application.

2. Zero-Knowledge Proof Systems

The concept of zero-knowledge was first introduced by three MIT researchers, Shafi Goldwasser, Silvio Micali and Charles Rackoff [35], where they were working on interactive proof systems in which the prover convinces the verifier that some statement holds by sending interactive messages. Previously, the research work in this context was assumed to have an honest verifier where a malicious prover tries to convince the verifier about the correctness of some statement. These researchers turned the problem and gave a new aspect in which a verifier can also be malicious. Precisely, they emphasised; how much extra information the verifier can derive from the proof transcripts other than the fact that the statement holds.

Any ZKP proof system must have the following three properties:

- **Correctness:** If the statement is true, the prover should be able to convince the verifier with overwhelming probability.
- **Soundness:** If the statement is false, the prover should not be able to convince the verifier at any cost.
- **Zero-Knowledgeness:** The verifier must not be able to learn anything except that the statement holds.

Proving correctness can be done easily by playing multiple rounds of the protocol interactively giving a probabilistic guarantee to the proof system. To prove soundness, we make use of the existence of a knowledge extractor that interacts with the prover and can extract the witness from the transcripts if the protocol is completed successfully. The fact that the extractor can retrieve the witness from transcripts implies that the witness was injected into the transcripts by the prover.

The challenging part comes in proving the last property. Researchers have argued that zero-knowledgeness can be proven by using the concept of Simulation. If it can be proven that there exists a simulator that has no information and whose transcript is identically distributed to the real prover, then the verifier can extract the same amount of knowledge from the real transcripts as can be extracted by simulated transcripts; however, as the simulated transcripts have no information in the first place, the verifier cannot extract any information from the real transcript as well.

2.1 Embedded Curves

Zk-Snarks uses many cryptographic primitives [2,6,7,8,12,17,30]. Besides, we discuss here the embedded curve used in Zokrates to link the identity with the prover [12].

In Zokrates, all arithmetic operations are defined on a finite field [30], specifically, a Galois Field, $\boldsymbol{GF(p^n)}$ with $\boldsymbol{n = 1}$. This means all operations are modulo $\boldsymbol{p}$ where $\boldsymbol{p}$ is the order of a group of elliptic curves [7]. In Zokrates, this $\boldsymbol{p}$ is defined as

$$\boldsymbol{p = 21888242871839275222246405745257275088548364400416034343698204186575808495617}$$

This value is taken so that, it is equal to the group order of the BN128 curve used in Ethereum. This makes verification on the blockchain much cheaper as Ethereum provides precompiled contracts for the BN128 curve. As elliptic curve operations such as addition and multiplication involve modular arithmetic and modulo operations are inefficient in SNARKs, incorporating elliptic curve cryptography becomes very expensive in the Zokrates system.

This is solved using an embedded curve in Zokrates, BabyJubJub, which has parameters such that the order of the field over which it is defined becomes equal to the group order of the system curve. This way elliptic curve operations get reduced to the simple field arithmetic in Zokrates and make elliptic curve operations nearly free.

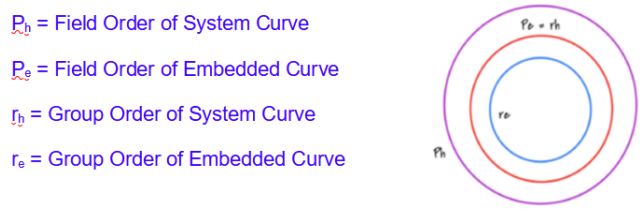

Figure 1: Embedded Curve

3. Understanding Zokrates

Zokrates is a toolbox that uses SNARKs for verifiable computations. It provides us with all the tools from specifying the constraints in DSL to export the verification code to Solidity smart contract. In this section, we discuss the details of the Pinocchio Protocol by PGHR13[26] and, finally, we discuss Zokrates.

3.1 Pinocchio Protocol

A verifiable computation contains three algorithms $(\boldsymbol{Setup}, \boldsymbol{Compute}, \text{and } \boldsymbol{Verify})$. $\boldsymbol{Setup}$ takes the computation function, a security parameter, and converts it to Common Reference String (CRS). This will output a proving and verification key. $\boldsymbol{Compute}$ will take the computation function, inputs and proving key and gives the output to computation and proof. $\boldsymbol{Verify}$ will verify the proof using the verification key. Proof needs to be zero knowledge for our case.

We consider four important aspects of this protocol.

- **Correctness:** For any function F and any input u, if we run $(\boldsymbol{EK_F}, \boldsymbol{VK_F}) \leftarrow (\boldsymbol{F}, \boldsymbol{1^\lambda})$ and $(\boldsymbol{y}, \boldsymbol{\pi_y}) \leftarrow \boldsymbol{Compute}(\boldsymbol{EK_F}, \boldsymbol{u})$, then we always get $\boldsymbol{1} = \boldsymbol{Verify}(\boldsymbol{VK_F}, \boldsymbol{u}, \boldsymbol{y}, \boldsymbol{\pi_y})$. Here $\boldsymbol{EK_F}$ and $\boldsymbol{VK_F}$ are the evaluation and verification keys. This comes from the completeness property of proof systems.
- **Security**: For any function $\boldsymbol{F}$ and any probabilistic polynomial-time adversary $\boldsymbol{A}$, $\boldsymbol{Pr}[(\boldsymbol{u}, \boldsymbol{y}, \boldsymbol{\pi_y}) \leftarrow \boldsymbol{A}(\boldsymbol{EK_F}, \boldsymbol{VK_F}) : \boldsymbol{F(u)} = \boldsymbol{y}$ and $\boldsymbol{1} = \boldsymbol{Verify}(\boldsymbol{VK_F}, \boldsymbol{u}, \boldsymbol{y}, \boldsymbol{\pi_y})]$ is negligible.
- **Zero-Knowledgeness:** If $\boldsymbol{F(u, w)}$ is a function with $\boldsymbol{u}$ as the public input and $\boldsymbol{w}$ as the private input, then given a proof $\boldsymbol{\pi_y}$ and output $\boldsymbol{y}$ for the given function $\boldsymbol{F}$, there must not be any way of extracting $\boldsymbol{w}$ from the given information.
- **Efficiency:** $\boldsymbol{Verify}$ must be cheaper as compared to $\boldsymbol{Compute}$. $\boldsymbol{Setup}$ is also important but this depends on the underlying constraints, so the amortised cost is reasonable.

KEA Assumption (Knowledge of Exponent Assumption): For any adversary $\boldsymbol{A}$, taking input $\boldsymbol{q}, \boldsymbol{g}, \boldsymbol{g^a}$ and returns $(\boldsymbol{X}; \boldsymbol{Y})$ with $\boldsymbol{Y} = \boldsymbol{X^a}$, there always exists a knowledge extractor $\boldsymbol{K}$ which given the same inputs as $\boldsymbol{A}$, returns $\boldsymbol{x}$ such $\boldsymbol{g^x} = \boldsymbol{X}$. Additionally, if given two points $\boldsymbol{A}$ **and** $\boldsymbol{B}$ where $\boldsymbol{B} = A^c$ and a point $\boldsymbol{P}$, then the only way to calculate $\boldsymbol{P^c}$ is when $\boldsymbol{P}$ is derived from $\boldsymbol{A}$; that is, there exists some $\boldsymbol{\gamma}$ that is $\boldsymbol{\gamma}.\boldsymbol{A} = \boldsymbol{P}$.

Quadratic Programs: Now, we assume an arithmetic circuit and define a Quadratic Arithmetic Program (QAP). For simplicity, we assume a simple circuit as shown in Figure 2 with four inputs and two outputs from multiplication gates. $\boldsymbol{p_1}$ and $\boldsymbol{p_2}$ are the inputs to gate $\boldsymbol{G_1}$. $\boldsymbol{p_3}, \boldsymbol{p_4}$ and $\boldsymbol{p_5}$ are the inputs to gate $\boldsymbol{G_2}$(addition gates are not considered). $\boldsymbol{p_5}$ and $\boldsymbol{p_6}$ are the outputs of gates $\boldsymbol{G_1}$ and $\boldsymbol{G_2}$, respectively.

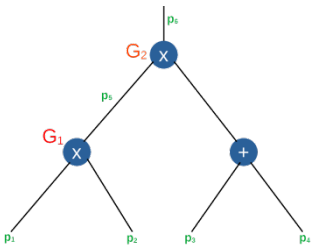

Figure 2: Circuit for QAP

QAP is defined as:

Q: Let $\mathbf{V} = \{\boldsymbol{v_k(x)}\}$,$\mathbf{W} = \{\boldsymbol{w_k(x)}\}$, $\mathbf{Y} = \{\boldsymbol{y_k(x)}\}$ for $\boldsymbol{k} \in \{\boldsymbol{0}..\boldsymbol{m}\}$ be three sets of $\mathbf{m+1}$ polynomials and $\boldsymbol{t(x)}$, a target polynomial. Let $\boldsymbol{F}$ be a function taking $\boldsymbol{n}$ elements of $\boldsymbol{F}$, giving $\boldsymbol{n'}$ outputs and let $\boldsymbol{N} = \boldsymbol{n} + \boldsymbol{n'}$. Then, $\boldsymbol{Q}$ computes $\boldsymbol{F}$ if $(\boldsymbol{p_1}, \boldsymbol{p_2}_{\ldots\ldots}\boldsymbol{p_n})$ is a legal assignment of $\boldsymbol{F}$, iff ∃ coefficients $(\boldsymbol{p_{N+1}} \ldots \boldsymbol{p_m})$ such that $\boldsymbol{t(x)}$ divides $\boldsymbol{p(x)}$. Here $\boldsymbol{p(x)}$ is defined as

$$\boldsymbol{p(x)} = \left(\boldsymbol{v_0(x)} + \sum_{\boldsymbol{k=1}}^{\boldsymbol{m}} \boldsymbol{c_k(x).v_k(x)}\right).\left(\boldsymbol{v_0(x)} + \sum_{\boldsymbol{k=1}}^{\boldsymbol{m}} \boldsymbol{c_k(x).w_k(x)}\right) - \left(\boldsymbol{y_0(x)} + \sum_{\boldsymbol{k=1}}^{\boldsymbol{m}} \boldsymbol{c_k(x).y_k(x)}\right)$$

The size of $\boldsymbol{Q}$ is $\boldsymbol{m}$ and degree is $\boldsymbol{degree(t(x))}$.

Now we select a root $r_g \in F$ for each multiplication gate and express the target polynomial $t(x)$ as $\prod_g(x - r_g)$. V, W **and** Y are defined such that V encodes the left input for each multiplication gate, W encodes the right input and Y encodes the outputs. Also, we define

$$v_k(r_g) = \begin{cases} 1, & k^{th} \textbf{ wire is a left input to gate } g \\ 0, & \textbf{otherwise} \end{cases}$$

$w_k(r_g)$ and $y_k(r_g)$ are defined in a similar way. Now if we look at a specific gate G_i and its root r_g. Equation 4.1 becomes

$$\left(\sum_{k=1}^{m} c_k(x).v_k(x)\right).\left(\sum_{k=1}^{m} c_k(x).w_k(x)\right)$$
$$= \left(\sum_{k \in I_{left}} c_k\right).\left(\sum_{k \in I_{right}} c_k\right)$$
$$= c_g.y_k(r_g) = c_g$$

which simply means that for any multiplication gate product of inputs is equal to the output.

Trusted Setup: We take KEA Assumption and extend it further by saying that if we have n pair of points (P_1, Q_1), $(P_2, Q_2) \ldots\ldots (P_n, Q_n)$, where $\forall i, P_i.k = Q_i$ and we need to come up with two points (P, Q) such that $P.k = Q$. Now if k is known, this becomes very trivial; therefore, k needs to be hidden or thrown out after using so that it cannot be used again. This dumping of toxic waste is important and the whole task of generating these points is known as a trusted setup and must only be performed by someone trustworthy. Considering this, the only way to come up with a point (P, Q) such that $P.k = Q$ is when P is a linear combination of $(P_1, P_2 \ldots P_n)$ and Q is a linear combination of $(Q_1, Q_2 \ldots Q_n)$ which implies that the coefficients are known by the prover.

Verifiable Computation: In a real-world scenario, most of the time the polynomials V, W and Y are very large; therefore, we cannot use them directly. To solve this problem, polynomials are converted into elliptic curve points. Using elliptic curve points also helps in verifying the correctness. Formally, instead of sending polynomials V,W and Y, we send elliptic curve points in the form:

- $G * v_1(t)$, $G * v_1(t) * k_v$
- $G * v_2(t)$, $G * v_2(t) * k_v$
- ……..
- $G * w_1(t)$, $G * w_1(t) * k_w$
- $G * w_2(t)$, $G * w_2(t) * k_w$
- ……..
- $G * y_1(t)$, $G * y_1(t) * k_y$
- $G * y_2(t)$, $G * y_2(t) * k_y$
- ……..

Here t, k_v, k_w **and** k_y are toxic wastes. Now assuming the extended KEA assumption, the prover needs to send the following values:

- $\pi_v = G * V(t), G * V(t) * k_v$
- $\pi_w = G * W(t), G * W(t) * k_w$
- $\pi_y = G * Y(t), G * Y(t) * k_y$

To make sure all these linear equations are using the same coefficients, this value is also added to the setup: $Q = G * (V(t) + W(t) + Y(t)) * b$. b is again the toxic waste. Then, we use elliptic curve pairings to verify that $V * W - Y = H - P$. We check that

$$e(\pi_v, \pi_w)/e(\pi_y, G) = e(\pi_h, G * P(t))$$

To check that all combinations are using the same coefficients, we again use the pairings and verify that Q matches with the provided $V + W + Y$.

3.2 Zokrates

Zokrates uses the idea of the delegation of computation. Computation is delegated to a single node rather than all nodes traditionally and that node executes the logic and publishes the result on-chain (Figure 3). This method gives two advantages.

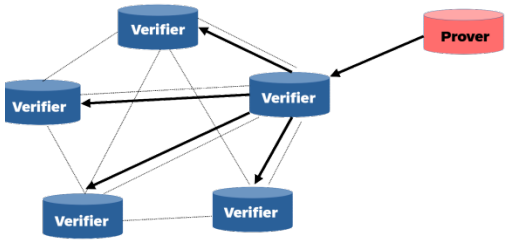

Figure 3: Delegated Computation in Zokrates

- The delegate node can use private information to execute the computation and publishes only the result. This is not possible in the traditional blockchain setting.
- Delegate Node only writes the result to the blockchain which increases efficiency in a way that all the nodes only store the result.

However, the problem here is any delegated node needs to be trusted. Therefore, the idea of verifiable computation is employed using Pinocchio Protocol. Delegated Node becomes the prover and computes the proof for computation, which is then verified by nodes on the blockchain. Privacy can be maintained by using zero-knowledge proofs.

3.2.1. Architecture

Zokrates supports writing the code in high-level language and converting it to a verification smart contract so that it can be

deployed and the proofs verified on-chain. It has some inbuilt components for its processes. Below is the summary of each component in Zokrates.

- **Compiler**: Parsing and Flattening of Code is done by the Compiler inside Zokrates. After flattening, the constraints are transformed into a format that can be easily converted into R1CS constraints.
- **Witness Generator:** Before executing the program and generating the proof, the code must be given a valid assignment of input variables. The witness generator takes the valid inputs, interprets the flattened code and generates the witness.
- **Circuit Importer:** Sometimes, flattened code is hand optimised by developers. The circuit importer supports the functionality of importing the constraints directly into the Zokrates toolbox.
- **Setup and Proof Generator:** Setup takes the code and witnesses generating an evaluation and verification key. These keys are used in proof generation and verification.
- **Contract Generator:** According to the verification key, a solidity contract is generated which has all support for ECC operations using bn256g2 library and for providing elliptic curve pairing operations in verifyTx method which is called to verify the transaction.

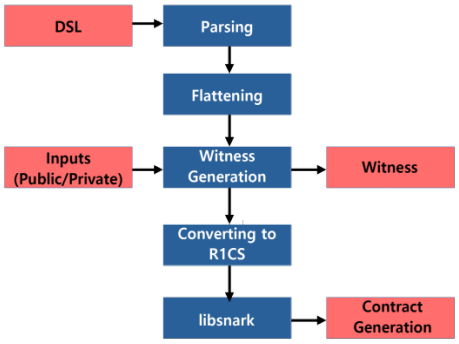

Figure 4: Zokrates Components

Zokrates internal processes are summarised in Figure 4. Zokrates can be used with three proving schemes currently, namely, PGHR13, Groth16 and GM17. In our application, we have mainly used PGHR13 and Groth16. Groth16 has some variations like shorter proof size (only 3 curve points are given as proof as compared to 8 in PGHR13) which makes it more efficient.

4. Problem Statement

Hedge Funds are more private investment firms. The fund manager after collecting the investment from all investors starts investing it. They use different strategies and statistical techniques to allocate the amount in different assets. This allocation is private to a firm and not disclosed by the fund managers as this might leak the strategies used by them. We define a set $\boldsymbol{A}$ containing all the assets in which a fund manager makes any investment.

$$\boldsymbol{A} = \{\boldsymbol{A_1}, \boldsymbol{A_2}.....\boldsymbol{A_n}\}$$

Such that $|A| = n \in Z$.

For any investor, his/her investment is allocated in different assets in A. We define these allocations by weight $\boldsymbol{w_i}$ (fraction of total investment assigned in a particular asset). These are also called portfolio weights. An allocation for an investor in different assets defines their portfolio. Portfolio weights are kept private by fund manager. Here $\boldsymbol{W}$ is the portfolio, $\boldsymbol{w_i}$ is the fraction of total investment invested in asset $\boldsymbol{A_i}$.

$$\boldsymbol{W} = \{\boldsymbol{w_1}, \boldsymbol{w_2}.....\boldsymbol{w_n}\}$$

Note that,

$$\sum_{i=1}^{n} \boldsymbol{w_i} = \mathbf{1}$$

An example of 3-fund portfolio (having only 3 assets) is:

Table 1: Example 3-Fund Portfolio

Asset(A)	Allocation (w_i)
U.S. 'Total Market' Index Fund	0.6
International Stock 'Total Market' Index Fund	0.3
Bond 'Total Market' Index Fund	0.1

Investors in these funds expect the higher returns but they also expect that amount of risk should not be too high. For example, investing too much of an investment amount in an asset that has a higher risk degree might introduce a conflict of interest with the investors. An investor might not be comfortable with too much amount assigned to a single asset. To estimate the risk for each asset in the market, fund manager calculates the risk factor $\boldsymbol{f_i}$. These quantities are public.

The fund managers need to convince the investor that they are following the guidelines and not investing too much of their money into a risky investment. So, the condition defined is

$$\boldsymbol{X} \leq \boldsymbol{t} = \sum_{i=1}^{n} \boldsymbol{w_i f_i} \leq \boldsymbol{Y}$$

where $\boldsymbol{X}$ and $\boldsymbol{Y}$ are the limits specified by investor.
Sometimes risk factors are specified as the correlation between any two assets such that $\boldsymbol{f_{i,j}}$ specifying the risk factor if both $\boldsymbol{A_i}$ and $\boldsymbol{A_j}$ are used in high or low proportion. Correspondingly, non-linear conditions can be defined as

$$\boldsymbol{X} \leq \boldsymbol{t} = \sum_{i=1}^{n} \boldsymbol{w_i w_j f_{i,j}} \leq \boldsymbol{Y}$$

Sometimes, the investor also wants portfolio weights not to exceed a certain quantity for a single asset. This gives us the following (individual condition):

$$\forall \boldsymbol{i} \in [\boldsymbol{1} \ldots \boldsymbol{n}] \quad \boldsymbol{w_i} \leq \boldsymbol{h}$$

where $\boldsymbol{h}$ is the individual risk threshold for each asset.

5. Protocol Workflow

In this section, we present a protocol to be used by the fund manager and investors that allows investors to be convinced that fund managers are behaving properly. After that, we discuss some implementation details.

5.1 Participants

- **Fund Manager/Prover:** Fund Manager needs to follow the protocol to convince the investor of specified conditions. (Or the Financial body may employ an auditor to accomplish this task of proving.)
- **Investor/Verifier:** Investors will give the conditions or agree upon predefined conditions, participate in the protocol and wait for the prover to convince him/her.
- **Government Regulatory Body:** Regulatory Body provides all the necessary guidelines that need to be followed by the fund manager/prover to avoid any conflict of interest with the investors and ensure transparency in some way.

5.2 Protocol

There are two phases in this protocol. Initial Phase and Use Phase.

5.2.1. Initial Phase

i. The fund manager will publish the details of portfolio characteristics including the universe of assets(A), risk factors(F) and the public key to be used for convincing the verifier. Investors will only invest if they agree upon these points.
ii. Fund Manager deploys Record contract and publishes the contract address and ABI.
iii. Investors register themselves on Record smart contract and send the obtained ID to the Fund Manager on a secured private channel confirming their participation.

5.2.2. Use Phase

i. Investors will compile the DSL specifying all conditions, the public key of the prover and export the verification smart contract by specifying their constraints.
ii. Investors deploy the contract on the blockchain, set the contract address and proving key hash on the Record smart contract.
iii. Investors can also provide their custom conditions and limits if agreed by the fund manager initially(optional).
iv. The prover/fund manager will compile the imposed DSL and make sure that the bytecode matches with the smart contract deployed. Prover, then computes the witnesses generating the proof in JSON format using their private key and proving key shared by the investor.
v. The prover will upload the proof as JSON and call the verifyTx method on the smart contract.
vi. Verifier will watch for Success Event on the smart contract deployed to get convinced that all the conditions are satisfied, and that proof was generated by the fund manager only. If the event is not triggered, investor can report to the regulatory body.

Figure 5 gives a basic illustration of the protocol.

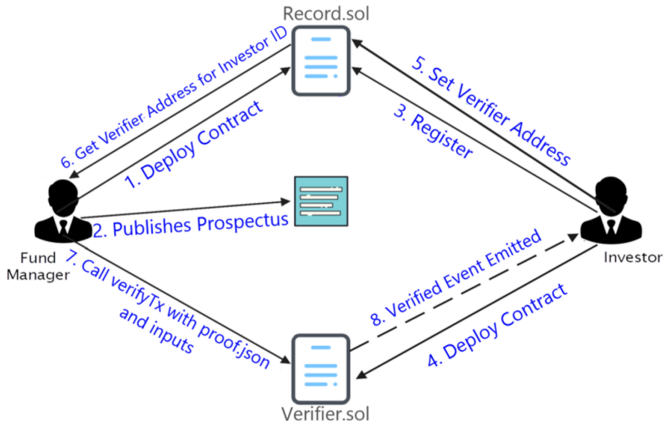

Figure 5: Protocol between Fund Manager and Investor

5.3 Implementation

The record contract deployed by the Fund Manager is written in pure Solidity. Full code can be found in Appendix A. The contract has three methods.

- **Register ():** This method is called by investors in Initial Phase. It generates a unique ID for each investor incrementally, stores the id in the mapping with the investor address and returns the ID.
- **Set ():** This method is also called by an investor in Use Phase to set the Verifier address and proving key for them. It also verifies that only investors should be able to call this method for themselves.
- **Get ():** This method is called by the Fund Manager in Use Phase. It returns the Verifier address and proving key for a given Investor ID. It also verifies that only the fund manager(owner) should be able to call this method.

The DSL for Zokrates is prewritten and contains values like the public key of the fund manager, risk factors and so on. Values like X, Y and h are injected by the investor before compiling. As proving key is very large, storing it on the smart contract is not viable, so investor first uploads the key file on IPFS and stores the obtained IPFS hash on Record Contract. The prover then retrieves it by the given hash. There are n+1 private arguments for n portfolio weights. One input is the private key generated

from BabyJubJub Curve. ECC library provides us with cryptographic support with Edwards Curve (embedded curve in Zokrates) which fits well within the context of Zokrates.

After compiling, constraints are converted to QAP and finally exported to Solidity smart contract. This contract is deployed on Ropsten Testnet by the investor sharing contract address and key hash on Record Smart Contract. The Fund Manager gets the contract address from the Record Contract. The Record Contract is compiled such that only the fund manager (owner) can get this data of investor and nobody else other than the investor can set their details like contract address etc. After getting the address, the fund manager computes the witnesses and generates the proof in the form of JSON which is used directly to call verifyTx function.

6. Evaluation and Results

In this section, we analyse and evaluate the processes involved in our protocol. We divide our evaluation into two parts: (1) On-chain verification and (2) off-chain processes like generating keys, generating proof, etc.

6.1 Verification on-chain

The most significant part of the protocol is on-chain verification. We performed our testing on Ropsten Testnet. As verifyTx method is dependent on proof and the number of public inputs, verification will take constant time in our application irrespective of the size of the asset list. Therefore, even with many constraints in our application, verification will always be efficient. We compared the verification for two protocols, PGHR13 and Groth16. As in Groth16, proof size is smaller as there are only three elliptic curve points, we found that Groth16 performance is better than PGHR13 with ≈ 0.2 million gas used in Groth16 as compared to ≈ 0.5 million in PGHR13. Also, in deployment, gas used by Groth16 is ≈ 0.9 million whereas, in PGHR13, it is ≈ 1.4 million. These values are the average of 20 transactions on Ropsten Network.

6.2 Off-chain Processes

Off-chain processes include compilation, key generation, exporting the verifier, computing witnesses and proof generation. PGHR13 scheme in Zokrates uses libsnark as its backend. Compilation and exporting the verifier are the core Zokrates processes while generating keys and proofs are done by libsnark in its components. First, we tested these steps using PGHR13 proving scheme on Zokrates and obtained the constraint system data for each number of assets.

Table 2: Constraints System Data in Zokrates

Assets #	Constraints #	Inputs(Private /Public) #	Variables #
10	17892	11	16005
20	29602	21	26144
50	64732	51	56565
100	123282	101	107265
200	240382	201	208665

Then to measure performance, we run a profiling routine for key-generation and proof generation on PGHR13 proving scheme using libsnark as given in [29] with the data obtained. This layout uses a dense synthetic R1CS structure, so all these results are the upper bound. For other processes like compilation, exporting the smart contract, and computing witnesses, we used time command on Linux Machine. Below is the data we obtained.

These results are calculated by taking the average runtime of 3 execution rounds for each step. From these results, we found that setup is the bottleneck for the verifier and takes most of the time.

Table 3: Profiling Results for Verifier

Assets #	Compile Time(X) (s)	Setup(Y)	Export-Verifier(Z)
10	0.069	6.816	0.009
20	1.427	11.342	0.011
50	2.519	20.600	0.063
100	4.961	51.536	0.509
200	7.717	97.342	0.143

Table 4: Profiling Results for Prover

Assets #	Compile Time(X) (s)	Compute-Witness(Y)	Proof-Gen(Z)
10	0.042	0.395	2.256
20	1.347	0.485	3.399
50	2.532	0.787	6.202
100	4.942	1.132	11.572
200	7.943	2.279	22.302

From the graph in Figure 6, we conclude that for a few hundred assets, the verifier can complete execution in approximately 8–9 minutes and the whole process can be completed in about a minute for a single investor.

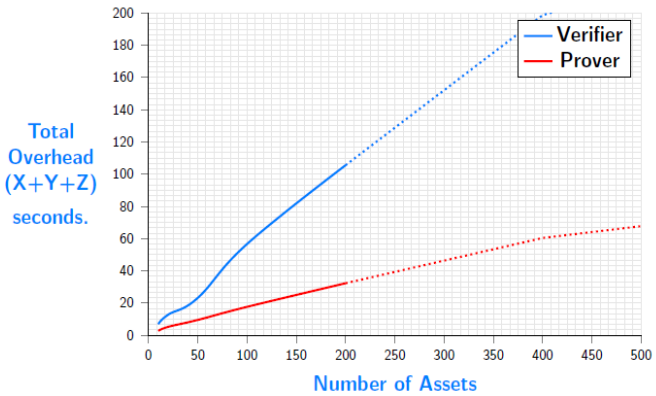

Figure 6: Total Overhead for Prover and Verifier

7. Conclusion

The protocol presented provides us with the ability to use zero-knowledge proofs in the financial regulatory system. Based on our implementation and analysis, we conclude that using Zokrates (or SNARKS) offers us a variety of ways to

come up with the compliance system. Using this, a lot of real-world bottlenecks like paper trails and account-keeping can be avoided. Also, as every financial organisation must be compliant with a regulatory body, such as SEC, this use-case serves as an introductory solution to many regulation environments.

7.1 Scope for Future Work

In our implementation, we have made some assumptions that can be handled to improve the application and explore some other opportunities. For example, we assumed the precision of up to 10 bits for weight quantities. This can be further extended if the number of assets is lower in number such that the resulting risk measure can fit well in Zokrates field type. Also, we can try other types of conditions which might be important from the financial point of view. In addition to this, we can also come up with a different protocol that uses other proving schemes like Bulletproofs integrated with some refereed delegation approach to make the verification cheaper.

Appendix A

A.1 Record Contract

```
---------------------------------
pragma solidity >=0.4.0 <0.7.0;
pragma experimental ABIEncoderV2;
contract Record {
uint ID;
address owner;
struct cust_type {
address addr;
bytes key;
}

mapping (uint => address) ad;
mapping (uint => cust_type) dta;
constructor () public {
owner = msg. sender;
ID = 0;

}
function register () public returns (uint){
uint t = ID;
ID=ID +1;
ad[t]= msg. sender;
return t;
}
function set (uint id, address sa, bytes memory ev_key) public {
assert (ad [id]== msg. sender);
dta [id]= cust_type ({
addr: sa,
key: ev_key
});
}
function get (uint id) public view returns (cust_type memory) {
assert (msg. sender == owner);
cust_type memory c = cust_type ({
addr: dta [id]. addr,
key: dta [id]. key
});
return c;
}
}
```

Competing Interests:
None declared.

Ethical approval:
Not applicable.

Author's contribution:
Komal Kalra is the main author responsible for writing the manuscript, collecting data, proofreading, etc.

Funding:
National Blockchain Project Funds by National Security Council Secretariat, Government of India.

Acknowledgements:
Not applicable.

References:

[1] (Last Amended, 2017). SEBI (Alternative Investment Funds Regulations,2012. https://www.sebi.gov.in/legal/regulations/apr-2017/sebi-alternative-investment-funds-regulations-2012-last-amended-on-march-6-2017-_34694.html.

[2] Benarroch, D. (2019). Diving into the zk-SNARKs Setup Phase. https://medium.com/qed-it/diving-into-the-snarks-setup-phase-b7660242a0d7.

[3] Blog (2004). Witness-Indistinguishability. https://www.isical.ac.in/~rcbose/internship/lectures2016/rt02feigeshamir.pdf.

[4] Blog (2018). Decentralized Applications dApps. http://blockchainhub.net/decentralized-applications-dapps/.

[5] Bootle, J., Cerulli, A., Chaidos, P., Groth, J., and Petit, C. (2016). Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In Fischlin, M. and Coron, J.-S., editors, Advances in Cryptology {EUROCRYPT 2016, pages 327-357, Berlin, Heidelberg. Springer Berlin Heidelberg.

[6] Buterin, V. (2016). snarks. https://medium.com/@VitalikButerin/quadratic-arithmetic-programs-from-zero-to-hero-f6d558cea649.

[7] Buterin, V. (2017a). Exploring Elliptic Curve Pairings. https://medium.com/@VitalikButerin/exploring-elliptic-curve-pairings-c73c1864e627.

[8] Buterin, V. (2017b). Zk-SNARKs: Under the Hood. https://medium.com/@VitalikButerin/zk-snarks-under-the-hood-b33151a013f6.

[9] Bunz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., and Maxwell, G. (2018). Bulletproofs: Short proofs for con_dential transactions and more. In 2018 IEEE.Symposium on Security and Privacy (SP), pages 315-334.

[10] Math.columbia.edu. 2021. [online] Available at: <http://www.math.columbia.edu/~rf/extensionfields>.

[11] Lexology.com. 2021. *Maintaining confidentiality in fund documents: a realistic expectation? | Lexology*. Available at: <https://www.lexology.com/library/detail.aspx?g=49d8

8904-352d-4ad7-8a33-a9534443158a#:~:text=Fund%20managers%20are%20typically%20keen,they%20disclose%20to%20potential%20investors.&text=Hedge%20fund%20managers%20and%20others,be%20reassured%20by%20the%20result>
[12] Deml, S. (2019). Efficient ECC in zkSNARKs using ZoKrates. https://medium.com/zokrates/efficient-ecc-in-zksnarks-using-zokrates-bd9ae37b8186.
[13] Eberhardt, J. and Tai, S. (2018). Zokrates - scalable privacy-preserving off-chain computations. In 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), pages 1084-1091.
[14] Ethereum (2019). Learn about Ethereum. https://ethereum.org/learn/.
[15] Feige, U. and Shamir, A. (1990). Witness indistinguishable and witness hiding protocols. In Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing, STOC '90, page 416{426, New York, NY, USA. Association for Computing Machinery.
[16] Flood, M. D., Katz, J., Ong, S. J. J., and Smith, A. (2013). Cryptography and the economics of supervisory information: Balancing transparency and confidentiality. Microeconomics: Asymmetric Private Information eJournal.
[17] Fondation, Z. (2017). zk-snarks. https://z.cash/technology/zksnarks/
[18] Gowravaram, N. R. (2018). Zero Knowledge Proofs and Applications to Financial Regulation. http://nrs.harvard.edu/urn-3:HUL.InstRepos:38811528
[19] Green, M. (2014). Zero Knowledge Proofs: An illustrated primer. https://blog.cryptographyengineering.com/2014/11/27/zero-knowledge-proofs-illustrated-primer/.
[20] Green, M. (2017). Zero Knowledge Proofs: An illustrated primer2. https://blog.cryptographyengineering.com/2017/01/21/zero-knowledge-proofs-an-illustrated-primer-part-2/.
[21] Groth, J. (2016). On the size of pairing-based non-interactive arguments. In Fischlin, M. and Coron, J.-S., editors, Advances in Cryptology {EUROCRYPT 2016, pages 305-326, Berlin, Heidelberg. Springer Berlin Heidelberg.
[22] JonathanKatz (2004). Witness-Indistinguishability. http://www.cs.umd.edu/~jkatz/gradcrypto2/NOTES/lecture21.pdf.
[23] Lutkebohle, I. (2008). BWorld Robot Control Software. http://aiweb.techfak.uni-bielefeld.de/content/bworld-robot-control-software/ [Online; accessed 19-July-2008].
[24] matter lab (2020). Awesome zero knowledge proofs (zkp). https://github.com/matter-labs/awesome-zero-knowledge-proofs.
[25] Menezes, A. (2005). An introduction to pairing-based cryptography.
[26] Parno, B., Howell, J., Gentry, C., and Raykova, M. (2013). Pinocchio: Nearly practical verifiable computation. In 2013 IEEE Symposium on Security and Privacy, pages 238-252.
[27] Pass, R. and Venkitasubramaniam, M. (2010). Private coins versus public coins in zero-knowledge proof systems. In Micciancio, D., editor, Theory of Cryptography, pages 588-605, Berlin, Heidelberg. Springer Berlin Heidelberg.
[28] scipr lab (2020a). C++ library for zkSNARKs. https://github.com/scipr-lab/libsnark.
[29] scipr lab (2020b). Profiling Libsnark. https://github.com/sciprlab/libsnark/tree/master/libsnark/zk_proof_systems/ppzksnark.
[30] Stanford (2014). Finite Fields. https://web.stanford.edu/class/ee392d/Chap7.pdf.
[31] Szydlo, M. (2005). Risk assurance for hedge funds using zero knowledge proofs. In Patrick, A. S. and Yung, M., editors, Financial Cryptography and Data Security, pages 156-171, Berlin, Heidelberg. Springer Berlin Heidelberg.
[32] Teutsch, J., Straka, M., and Boneh, D. (2019). Retrofitting a two-way peg between blockchains.
[33] Zokrates (2019). Zokrates. https://github.com/Zokrates/ZoKrates.
[34] SEC.gov | Hedge Funds – A New Era of Transparency and Openness
[35] S Goldwasser, S Micali, and C Rackoff. 1985. The knowledge complexity of interactive proof-systems. In Proceedings of the seventeenth annual ACM symposium on Theory of computing (STOC '85). Association for Computing Machinery, New York, NY, USA, 291–304. DOI: https://doi.org/10.1145/22145.22178
[36] Damgård, Ivan & Nielsen, Jesper. (2008). Commitment Schemes and Zero-Knowledge Protocols (2007). Lecture Notes in Computer Science - LNCS.
[37] NTL: A Library for doing Number Theory (libntl.org)
[38] https://www.sec.gov/news/statement/aguilar-effective-regulatory-oversight-and-investor-protection.html

