PYRAMID: A Protocol for Private and Trustless Multi-level Marketing on the Blockchain (Long Version)

Giovanna Kobus Conrado, Amir Kafshdar Goharshady, Kha Nhat Long Nguyen

▶ **To cite this version:**

Giovanna Kobus Conrado, Amir Kafshdar Goharshady, Kha Nhat Long Nguyen. PYRAMID: A Protocol for Private and Trustless Multi-level Marketing on the Blockchain (Long Version). 2024. hal-04678612

HAL Id: hal-04678612
https://hal.science/hal-04678612v1

Preprint submitted on 27 Aug 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

PYRAMID: A Protocol for Private and Trustless Multi-level Marketing on the Blockchain (Long Version)

Giovanna Kobus Conrado
Department of Computer Science
HKUST
Clear Water Bay, Hong Kong
gkc@connect.ust.hk

Amir Goharshady
Department of Computer Science
HKUST
Clear Water Bay, Hong Kong
goharshady@cse.ust.hk

Kha Nhat Long Nguyen
Department of Computer Science
HKUST
Clear Water Bay, Hong Kong
knlnguyen@connect.ust.hk

***Abstract*—The inherent traceability of blockchain transactions and smart contracts raises serious privacy concerns, especially in contexts such as private multi-level marketing in which maintaining the anonymity of relationships is paramount. In this work, we present PYRAMID, a protocol leveraging blind signature techniques that provides a smart-contract-based multi-level marketing implementation which is both private, i.e. no one can find out the relationships between people, and trustless. We specifically focus on the unique challenges posed by pyramid-like schemes, where funds must flow between members and their referrers without revealing their relationship. Our protocol achieves this by anonymizing the referral process, allowing for secure transactions without compromising privacy. We also provide a detailed security analysis showing that our approach attains the desired operational and privacy requirements.**

***Index Terms*—Multi-level Marketing, Blockchain, Smart Contract Applications**

I. Introduction

Blockchain and Smart Contracts. Pioneered by Bitcoin [1], Blockchain protocols provide a general solution to the problem of decentralized consensus. In a Blockchain-based system, all honest nodes of the network can reach consensus about an ordered set of transactions. In Bitcion, these transactions are simply transfers of money. However, later protocols such as Ethereum [2] built upon the groundbreaking idea that transactions can perform any well-defined and unambiguous computation. This led to the so-called *programmable* blockchains, which form the vast majority of new protocols in the past decade, and on which transactions can not only transfer money or tokens, but also deploy a piece of code (a smart contract) or interact with already-deployed smart contracts, for example by calling their functions. Since we have consensus over the transactions and their order, any node on the network can simulate the smart contracts and thus we will reach consensus about the state of every contract and the balance of every account. The decentralized and trustless nature of blockchain made smart contracts the ideal platform for a wide variety of financial contracts, such as auctions [3], escrows [4], [5], credit reporting [6], games [7]–[9], distributed problem-solving [10], elections [11], [12], random number generation [13]–[18] and loans [19].

Multi-level Marketing [20]. In this work, our focus is on a specific real-world type of multiparty financial contract, i.e. Multi-Level Marketing (MLM). An MLM starts with a finite set of founders, who are also members. Each member can recruit new members and become their "referrer". Each new member pays a membership fee to the system. The members also pay commercial revenues to the network from time to time. Part of these funds goes to the member's referrer, the referrer's referrer and so on. MLMs are also known as pyramid schemes. For our technical purposes in this work, there is no distinction between the two terms. However, there is a legal distinction which is orthogonal to our work: MLMs are legal in most jurisdictions and their main source of income is not the membership fees but the sale of a product by the members to the general public. Thus, they have independent commercial income. On the other hand, pyramid schemes are illegal in most jurisdictions, considered a scam, and merely function by transferring membership funds from new recruits to the top of the pyramid without providing any separate good or service.

The Challenge. As in the many other financial contracts named above, it would be desirable to implement an MLM as a smart contract and provide a decentralized and trustless system in which everyone is assured that they will receive their portions of the revenue. The primary challenge is that blockchains and smart contracts are transparent by design. Everyone can see the public list of transactions, which in turn shows the flow of money and all function calls to the contracts. Thus, if implemented naïvely, an MLM smart contract will leak the relationships between the members and anyone on the blockchain can know the referrer and referees of every member.

Our Contribution. In this work, we provide a novel MLM protocol called PYRAMID, using ideas and techniques from blind signatures and mixing, that not only achieves the benefits of a smart contract such as decentralization and trustlessness, but also provides strong privacy guarantees ensuring that no

other member or network node can unmask the referrer-referee relationship of any pair of members. Our protocol is implemented as an Ethereum smart contract. Interactions with smart contracts incur gas costs, minimizing which is an active area of research [21]–[23]. We show that our approach is also gas-efficient and highly cost-effective in practice.

II. Related Works

Our **PYRAMID** protocol is mainly based on blind signatures [24], a cryptographic primitive which we will formally define in Section III. Blind signatures have also been used for the following related purposes:

Auctions. In Decentralized Finance (DeFi) literature, many protocols have been proposed to address the challenges of conducting secure and transparent auctions [25], [26]. Many cryptographic primitives like Zero-knowledge proofs [25], [26], Multi-party computation [25], public-key encryption [26], and blind signatures [27] are used by these auction protocols.

E-Cash. Blind signatures were originally designed to enable the e-cash protocol [24]. This protocol predates blockchains and uses blind signatures to provide electronic money with privacy and unlinkability between the payer and the payee [24], [28]. This can be seen as a digital counterpart of anonymous transactions in physical currency. In e-cash, users withdraw electronic coins from a bank and spend them with merchants. Later, each merchant deposits the collected coins back to the bank. However, the bank is unable to connect the deposits to the withdrawals and thus has no information regarding which user paid which merchant.

Voting. The transparency provided by blockchain is highly useful for electronic voting. In this context, blind signatures have been extensively employed in the design of secure voting protocols on the blockchain, ensuring both transparency of outcome and user privacy, i.e. guaranteeing that no one can know which user voted for which candidate [28]–[32].

Coin Mixing. A mixer is a cryptographic protocol used to enhance privacy and anonymity in cryptocurrency transactions on a blockchain [33]. It works by combining and redistributing coins from multiple users, making it impossible to trace transactions back to their original senders. Coin mixers help protect user privacy by obfuscating transaction histories and preventing straightforward analysis of blockchain data. Many practical coin-mixing protocol have been developed using both blind signatures and other techniques [33]–[37]

III. Preliminaries on Blind Signatures

In this section, we cover the blind signature background needed for our approach. We refer to [24], [38] for a more detailed treatment. Blind signatures enable a user to obtain a valid signature on a message without revealing the message itself to the signer. The process typically involves the user "blinding" the message before presenting it to the signer, who then signs the blinded message. Upon receiving the signature, the user can "unblind" the signature on the blinded message to obtain a valid signature on the original message. In **PYRAMID**, we opt to use Schnorr Blind Signatures (SBS) as in [38]. We note that [39] showed Schnorr Blind Signatures are not secure under the ROS assumption. However, this was alleviated by [38]. In this work, we use blind signatures and partial blind signatures in a modular manner. Thus, to understand our approach, one can ignore the mathematical details of the construction below and focus only on the high-level guarantees provided by these signatures, i.e. the inability of the user, Alice, to forge signatures and the impossibility for the signer, Bob, to find any information about the message m. Nevertheless, we include all the details in order to be self-contained.

Schnorr Blind Signatures [38]. Suppose that Alice wants to obtain Bob's signature on a message m which must be kept private and not disclosed to Bob. SBS solves this problem in the following phases:

Setup Phase. The following parameters are chosen by Alice and Bob:

- A large prime number p,
- A multiplicative generator (primitive root) g which has order $q = p - 1$ modulo p,
- A cryptographic hash function H.

Additionally, Bob generates a private key x and computes and announces his public key $X = g^x \bmod p$.

Signing Phase. This phase starts with Alice communicating to Bob that she wants a blind signature on an undisclosed message. Let this message be m. They then take the following steps:

- *Nonce Generation*: Bob selects two random values $a, y \in \mathbb{Z}$ and computes

$$A = g^a$$

$$Y = X^y$$

  He sends A and Y to Alice. This effectively commits him to a and y without disclosing them.
- *Blinding*: Alice generates three random values $r_1, r_2, \gamma \in \mathbb{Z}_p$ and computes:

$$\begin{aligned} Y_0 &= Y^\gamma \\ A_0 &= g^{r_1} \cdot A^\gamma \cdot Y_0^{r_2} \\ c_0 &= H(A_0 || Y_0 || m) \\ c &= c_0 + r_2 \end{aligned}$$

  Intuitively, c is a blinded version of c_0. She sends c to Bob.
- *Signing*: Bob receives c from Alice and computes the signature s such that $s = a + c \cdot y \cdot x$ and sends (s, y) to Alice.

Unblinding Phase. Upon receiving (s, y) from Bob, Alice checks that $y \neq 0$, $Y = X^y$ and $g^s = A \cdot Y^c$. If these checks do not pass, she rejects the signature (s, y). Otherwise, Alice computes $s_0 = \gamma \cdot s + r_1$ and $y_0 = \gamma \cdot y$. The tuple (c_0, s_0, y_0) is the valid unblinded signature on m.

Signature Verification. The verification consists of checking that $y_0 \neq 0$, letting $Y_0 = X^{y_0}$ and $A_0 = g^{s_0} \cdot Y_0^{-c_0}$. The signature is valid if $c_0 = H(A_0||Y_0||m)$.

In this example, Alice successfully obtains a blind signature (c_0, s_0, y) from Bob for her message m without ever disclosing m to Bob or enabling him to find any information about m.

Partial Blind Signatures [38]. Partially-blind signature schemes extend blind signature schemes by enabling signers to include specific public information η, such as an expiration date, in the resulting signatures based on prior agreement with the receiver. This enhancement allows for greater flexibility and customization in the signed documents while preserving the privacy and security provided by blind signatures. In this work, we use the implementation of Partial Blind Signatures from [38] which is secure without relying on the ROS assumption. Suppose that Alice wants to obtain a signature on the undisclosed message m but the signature should also certify public information η. The protocol consists of the following phases:

Setup Phase. The following parameters are chosen by Alice and Bob:

- A large prime number p,
- A multiplicative generator (primitive root) g which has order $q = p - 1$ modulo p,
- Two cryptographic hash functions H and F.

As before, Bob generates a private key x and computes and announces his public key $X = g^x \bmod p$.

Signing Phase. This phase starts when Alice communicates to Bob that she wants a blind signature on an undisclosed message m. We also assume that Alice and Bob already agree on the public information η which should be embedded into the signature. They take the following steps:

1) *Nonce Generation*: Bob selects three random values $a, t, y \in \mathbb{Z}_p$ and computes $Z = F(\eta)$ and the commitments $A = g^a$ and $C = g^t \cdot Z^y$. He sends A and C to Alice.
2) *Blinding*: Alice receives (A, C) and computes $Z = F(\eta)$. She generates four random values $r_1, r_2, \gamma_1, \gamma_2$ and computes:

$$\begin{aligned} A_0 &= g^{r_1} \cdot A^{\gamma_1/\gamma_2} \\ C_0 &= C^{\gamma_1} \cdot g^{r_2} \\ c_0 &= H(\eta||A_0||C_0||m) \\ c &= c_0 \cdot \gamma_2 \end{aligned}$$

   As before, the intuition is that c serves as a blinded version of c_0. Alice sends c to Bob.
3) *Signing*: Bob checks that $c \neq 0$, computes $s = a + c \cdot x \cdot y$ and then sends (s, y, t) to Alice as signature

Unblinding Phase. Upon receiving (s, y, t) from Bob, Alice checks that $y \neq 0$, $C = g^t \cdot Z^y$ and $g^s = A \cdot X^{c \cdot y}$. She rejects the signature if these checks fail. If they pass, Alice computes $s_0 = (\gamma_1/\gamma_2) \cdot s + r_1$ and $y_0 = \gamma_1 \cdot y$ and $t_0 = \gamma_1 \cdot t + r_2$. The tuple (c_0, s_0, y_0, t_0) is the unblinded signature.

Signature Verification. The verification step checks that $y_0 \neq 0$ and computes $C_0 = g^{t_0} \cdot Z^{y_0}$ and $A_0 = g^{s_0} \cdot X^{-y_0 \cdot c_0}$. The signature is valid if $c_0 = H(\eta||A_0||C_0||m)$.

Informally, the approach is very similar to blind signatures, except that the public information η forms part of c_0.

IV. Design Goals and System Model

In this section, we give an overview of our **PYRAMID** protocol, its design goals and the security model.

A. Desired Functionality

The protocol is initialized with a set of *founding members*. An existing member Andy can refer a new member Bernie to join the protocol. We will call Andy the *parent* of Bernie. When Bernie joins, he must pay a protocol fee, that goes to the platform owner, and a joining fee, that goes to his parent Andy. Before accessing any received money, any non-founding members need to send a fixed percentage π of that money to their parents. In this case, if Andy is not a founding member, he must send a portion π of the money received from Bernie to his own parent and only then can Andy access the remainder. Similarly, if Andy's parent is not a founding member, they must send a portion π of the money received from Andy to their parent, and so on.

B. Security Model

We make the following standard security assumptions which are supported by the cryptographic primitives we will use in the design of our protocol:

Adversary Model. We consider an adversary that can control any number of accounts/members in the protocol, which may also include the signer.

Rational Adversary Assumption. The adversary will not perform an attack that causes them to lose money. We use a similar analysis of a rational adversary as in [40].

Properties of Blind Signatures. We assume the standard security properties of (partially) blind signature schemes:

- *Unforgeability*: It is impossible for an adversary to generate $k + 1$ valid message-signature pairs after observing k completed interactions with an honest signer.
- *Blindness*: It is infeasible for a dishonest signer to decide which of two messages m_0 and m_1 has been signed first in two executions with an honest user. This holds even if the signer is allowed to choose the public key.

C. Desired Security Guarantees

Our protocol must satisfy the following desired security requirements:

- *Theft-resistance*: If by honestly following the rules of protocol the adversary will gain m_0 units of money, then they should not be able to gain more than m_0 units by any dishonest behavior.
- *Unlinkability of Referrals*: An existing member Andy should be able to recruit a new member Bernie without the adversary being able to uncover their relationship, i.e. that Andy is Bernie's parent.

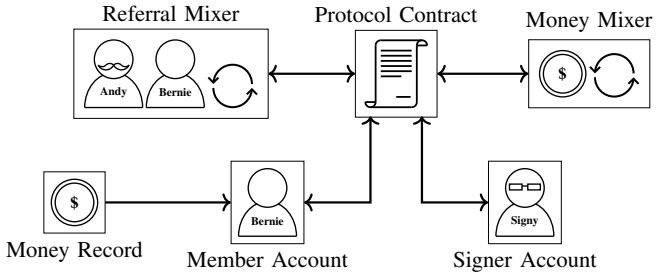

Fig. 1. System Architecture of the **PYRAMID** Protocol

- *Unlinkability of Money Flow*: Members should be able to transfer money *to their parents* without the adversary being able to uncover their relationship. Since we use blind signatures for our money transfers (see below), given the control of the signer account, the adversary is successful if they can link a blind signature with the unblinded signature, which corresponds to linking a member with their parent.

We use mixers to create both types of unlinkability. In doing so, we can inherit all the security guarantees of mixers as in [33]. In particular, this guarantees correctness in the amount of money sent and received from and to each member which is our theft-resistance property above.

V. Our Protocol

In this section, we first provide an overview of our protocol's system architecture and then present all the details.

A. System Architecture

The architecture of our protocol, as shown in Figure 1, consists of the following parts:

Protocol Contract. The protocol will be initialized with the following public parameters:

- Two hash functions: $H, F : \{0,1\}^* \to \{0,1\}^C$. The constant C depends on the setting in which the protocol is implemented. In practice, we use $C = 256$.
- A large prime number p.
- An integer g which is a primitive root modulo p.
- A portion $\pi \in [0,1]$, which models the percentage of money that a member needs to send to their parent before they can access the remainder.
- Three fixed fees: FEE_{PROT}, the protocol fee, FEE_{JOIN}, the joining fee that new members need to send to their parent after they have been referred, and FEE_{BID}, the bidding fee required for a member to become the signer. The role of the signer will become apparent further below.
- A fixed value μ which will be used in the money mixer.

These parameters will be the same for all rounds. Every pair of private and public keys in the protocol will be of the form (X, x) with the private key x and public key $X = g^x \bmod p$. The protocol will also record a list of blockchain addresses (ADDR) of member accounts.

Member Accounts. This is a new type of account, itself implemented by a smart contract. As a deployed smart contract, each member account will have a public address denoted as ADDR. Naturally, each member account is associated with one of the members in the protocol. The member account also contains two different public keys:

- *Public Send Key* (PUSK): This is a public key that is used to generate a signature to initiate a send transaction from the account. The signature needs to be generated using a corresponding private send key (PRSK) that is known only to the ***parent*** of the member owning this account.
- *Public Receive Key* (PURK): This is a public key that is used to initiate a receive transaction by the account. The signature needed to initiate a receive transaction will be created using a corresponding private receive key (PRRK) that is known by the member owning this account.

In the case of our founding members, who naturally have no parent, the public send key will be 0. Each member account will also hold a *money record* and contain further data (see below) supporting message signing.

Signer Account. The signer account is simply a member account that is selected as the signer in our current round. The additional information saved in the member account to support signing is:

- *Public Sign Key* (PUSIGN): This is the public key that is used to blind-sign a transaction when this member is the signer. The corresponding *Private Sign Key* (PRSIGN) is known by the member and satisfies $\text{PUSIGN} = g^{\text{PRSIGN}}$.
- *Signer Index* (SI): This indicates the priority of the member when they request to become a signer. The smaller the number, the higher the priority. This value will increase each time the member successfully becomes the signer and gets arbitrarily large if cheating behavior is detected.

Money Record (MR). The ownership of money by members will be encoded through a *money record*. Thus, each member account will also contain the following information:

- *Number of MRs*: A count of the MRs generated by this account through transfer activities.
- *List of MRs*: All MRs this account holds.

Each MR possesses, as attributes, the amount of money it holds, and a state which can be either "locked" or "unlocked". Locked MRs require the member to send π percent of their value to the member's parent in order to get unlocked. Unlocked MRs can be converted into the base cryptocurrency and withdrawn from the **PYRAMID** system. At any time, every member has the option of merging several of their locked MRs into a single locked MR whose value is the sum of the original values. This is implemented as a contract function. Similarly, they always have the ability to break down a locked MR into several smaller locked MRs whose total value is equal to the initial MR's value.

Referral Mixer. To support the unlinkability of referral of a new member, we utilize a referral mixer where we mix referrals in the same round with each other, making them indistinguishable. The mixer will record the following information:

- *Blind Schnorr Signature*: We use Blind Schnorr Signatures to implement the referral mixer. Thus, our implementation contains a function to verify such signatures.
- *List of Referral Requests*: This contains all the referral requests taking place in the current round.
- *List of Signatures*: This contains all signatures corresponding to the requests.

Money Mixer. To support the unlinkability of money flow from a member to their parent, we use a money mixer where MRs of a fixed value μ are mixed with each other. The mixer will have a send phase, receive phase, and verify phase, similar to [36]. The Mixer will record the following information:

- *List of Send Requests*: All send requests in the current round.
- *List of signatures*: All signatures corresponding to the referred requests. These are signatures created using members' PRSIGN.
- *List of receive requests*: All receive requests in the current round.
- Total amount of transferred money in this round.

Given the structure above, our protocol is comprised of two distinct workflows: one for the entry of new members into the system and another for the secure distribution of funds from a member to their parent.

Entry Workflow. The protocol employs a blind signature scheme for the entry of new members into the system. This mechanism ensures that the relationship between the referrer and the new member remains confidential. The blind signature process allows the referrer to obtain a signature on behalf of the new member without revealing the identity of the new member to any other party.

Distribution Workflow. To facilitate secure transactions between members and their parents, the protocol utilizes a partial blind signature scheme. This ensures that the total amount of money being transferred is publicly known while still maintaining the anonymity of the parties involved in each transaction.

Prevention of Cheating. The protocol works similarly to a mixer in both workflows: we employ a mechanism where the role of the central signer is rotated among member accounts. By employing game theory principles, the protocol aligns the incentives of internal members with the overall integrity of the system. Participants are incentivized to follow the protocol rules and act honestly, as deviations from the prescribed behavior would result in unfavorable outcomes for themselves.

B. Detailed Protocol Description

We will now detail the steps the protocol takes for deployment and the entry and distribution workflows.

Protocol Deployment. First, we will deploy the protocol contract, with all its initial parameters as previously defined.

Account Deployment. Each person who wants to join the protocol will need to deploy a smart contract following the member account model. This person will then acquire a public address for the member account. They should use this public address to join the protocol. The protocol will check that the public address corresponds to a member account.

Protocol Setup. This phase sets up the founding members. Each founding member will need to deposit an amount of money to enter the protocol. These members will have 0 as their PUSK and they will be able to unlock their MRs without the need for a transfer to parents.

Signer Rotation. The protocol works in rounds, and each round will have a member chosen as the signer. Each member can bid to become the signer of the next round during the time of the current round by depositing FEE_{BID}. When a member joins the protocol, their SI is set as the position they join the protocol, i.e. the k-th member will have $\text{SI} = k$. When a member successfully becomes the signer, their SI will increase by the number of current members in the protocol. When two members with the same smallest SI bid for the signer, the first come first serve rule will be applied. The remaining two workflows only happen after the protocol setup and registration of founding members. Round 0 is used to find the first signer, so all other actions start at round 1.

Member Referral. We now describe the workflow for the entry of non-founding (referred) members. Please see Figure 2. This follows the blind signature protocol in Section II. A person who wants to join the protocol must first deploy a member account and acquire a public address. The account will become a member of the protocol when the protocol contract records that address as a member. Suppose that Andy, which is already a member account of the protocol, wants to refer an account Bernie, which is currently not a member and that member Signy is the current signer. The steps are:

1) Andy requests a new referral code from Signy. In this step, Signy will choose $a, y \xleftarrow{\$} \mathbb{Z}_p$ and send $A = g^a$ and $Y = \text{PUSIGN}^y_{\text{Signy}}$ to Andy. The pair (A, Y) will also act as the nonce to identify the request.
2) Andy chooses three random numbers $r_1, r_2, \gamma \xleftarrow{\$} \mathbb{Z}_p$ and computes:

$$\begin{aligned} Y_0 &= Y^\gamma \\ A_0 &= g^{r_1} \cdot A^\gamma \cdot Y_0^{r_2} \\ c_0 &= H(A_0 || Y_0 || \text{ADDR}_{\text{Bernie}}) \\ c &= c_0 + r_2 \end{aligned}$$

   Andy sends c to Signy. Note that in this case, the secret message m which should not be disclosed to Signy or anyone else is simply Bernie's address. Moreover, Andy would agree to do this step only if he has access to the PRSK corresponding to the PUSK of Bernie's member account contract.
3) Signy checks that $c \neq 0$ and then computes $s = a + c \cdot \text{PRSIGN}_{\text{Signy}} \cdot y$ and then sends (c, s, y) to Andy as signature. All announcement are on-chain and recorded in the smart contract.
4) The smart contract checks that $y \neq 0$, $Y = \text{PUSIGN}^y_{\text{Signy}}$ and $g^s = A \cdot Y^c$. If these checks do not pass, it refuses to record the values provided by Signy. Not providing a

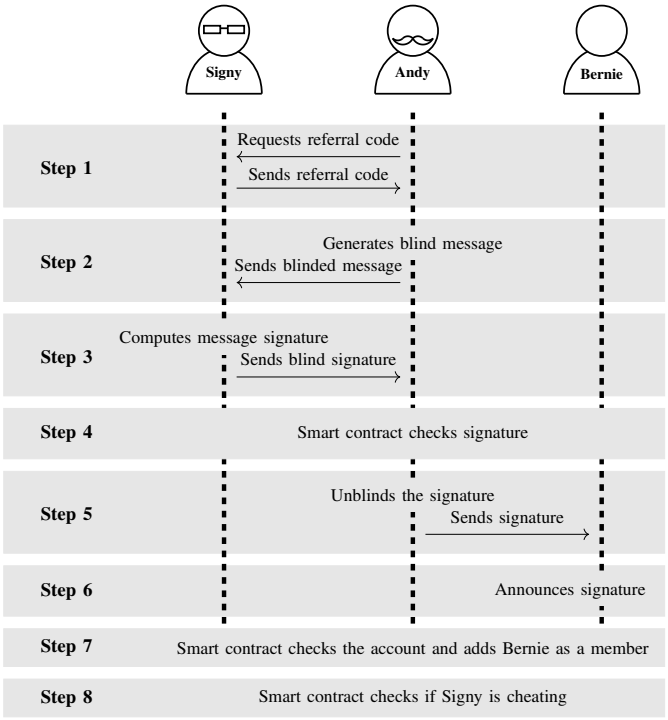

Fig. 2. **PYRAMID** Member Referral Workflow. Messages have been abstracted. In implementation, every communication other than the one between Andy and Bernie is done through the contract protocol, which performs the required checks automatically.

correct signature is seen as dishonest behavior by Signy and will be punished.

5) Andy computes $s_0 = \gamma \cdot s + r_1$ and $y_0 = \gamma \cdot y$. The tuple (c_0, s_0, y_0) is a valid signature. Andy sends this to Bernie.
6) After all the new registrations in the current round have passed the previous step, Bernie announces (c_0, s_0, y_0) for verification to the smart contract. He also pays the fees (see further below for details).
7) The verification step checks that $y_0 \neq 0$ and computes $Y_0 = \text{PUSIGN}_{\text{Signy}}^{y_0}$ and $A_0 = g^{s_0} \cdot Y_0^{-c_0}$. The signature is valid if $c_0 = H(A_0||Y_0||\text{ADDR}_{\text{Bernie}})$. After successful verification and checking that $\text{ADDR}_{\text{Bernie}}$ is a valid instance of the member account contract, the protocol contract registers it as a member.
8) The contract checks whether the number of added members in the current round is less than or equal to the number of referral requests in Step 1. If there are more members added than referred, then Signy is cheating. The contract will cancel all the referrals of the current round and punish Signy.

Initial Setup for New Member. As mentioned above, each newly-referred member needs to deposit FEE_{PROT} to join the protocol. The new account Bernie will need to send FEE_{JOIN} to the account Andy that referred them to the protocol. This money will become a locked MR that belongs to Andy. This is not applicable to the founding members since they are not referred and have no parents. To achieve privacy for this transfer, we utilize the following money flow.

Money Flow. We now describe how money flows between members and their parents. This is illustrated in Figure 3. We focus on the benefits associated with referrals, but one can transfer any other commercial income in the same manner. This workflow follows the partial blind signature protocol in Section III. Assume a member Bernie has to send money to their parent Andy and the current signer is a member Signy. Recall that the private send key PRSK corresponding to the PUSK recorded in Bernie's member account is only known to the parent Andy. Let Bernie have an MR with the value μ and suppose the current round number is r_{CURR}. The process is as follows:

1) Bernie initiates a send transaction.
2) Signy generates three random values $a, t, y \xleftarrow{\$} \mathbb{Z}_p$, computes $Z = F(\mu||r_{\text{CURR}})$ and then computes the commitment $A = g^a, C = g^t \cdot Z^y$. She sends (A, C) to Bernie via the smart contract.
3) Bernie receives (A, C), generates random values $r_1, r_2, \gamma_1, \gamma_2$ and computes:

$$\begin{aligned} A_0 &= g^{r_1} \cdot A^{\gamma_1/\gamma_2} \\ C_0 &= C^{\gamma_1} \cdot g^{r_2} \\ c_0 &= H(Z||A_0||C_0||\text{ADDR}_{Andy}) \\ c &= c_0 \cdot \gamma_2 \end{aligned}$$

   Here c_0 is the original message and c is the blind message.
4) Bernie sends all the values above to Andy and asks Andy to sign c using the PRSK corresponding to the PUSK in Bernie's member account.
5) Andy verifies that all values, especially c_0, are well-formed as above, and only if so, signs c. This ensures that Bernie cannot send money to anyone other than his parent, Andy, since if c_0 does not reference ADDR_{Andy}, he would simply refuse to sign it.
6) Bernie sends c to Signy over the smart contract. He also sends the signature obtained in the previous step. The contract verifies the signature using the PUSK in Bernie's account.
7) Signy checks that $c \neq 0$ and then computes $s = a + c \cdot \text{PRSIGN}_{\text{Signy}} \cdot y$ and sends (s, y, t) to Bernie and Andy as signature. This is also over the smart contract and visible to everyone. The smart contract checks the validity of the blind signature, i.e. it checks that $y \neq 0$, $C = g^t \cdot Z^y$ and $g^s = A \cdot \text{PUSIGN}_{\text{Signy}}^{y \cdot c}$.
8) After all transfers of the current round have passed the previous step, Andy initiates a receive transaction by providing a valid receive signature generated by $\text{PRRK}_{\text{Andy}}$. He reads (c, s, y, t) from the contract and computes

$$\begin{aligned} s_0 &= (\gamma_1/\gamma_2) \cdot s + r_1 \\ y_0 &= \gamma_1 \cdot y \\ t_0 &= \gamma_1 \cdot t + r_2 \end{aligned}$$

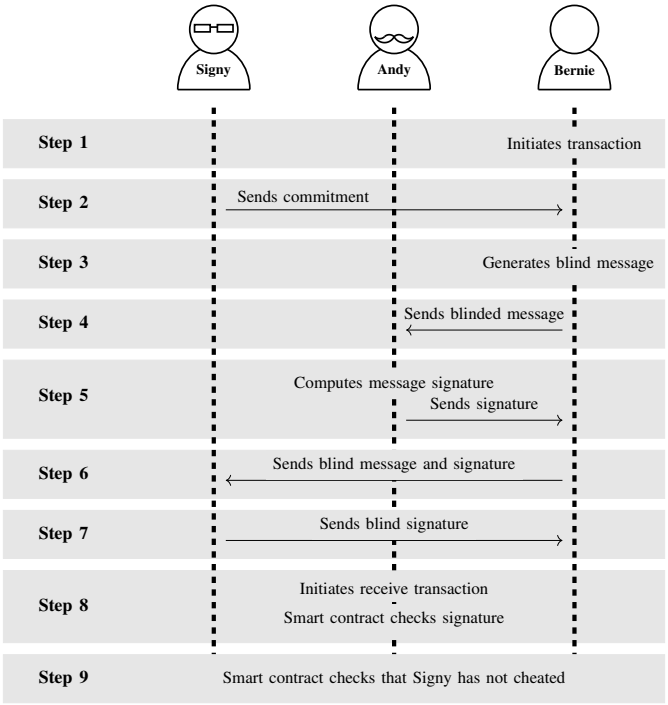

Fig. 3. Flow of Money from a Member Bernie to his Parent Andy in **PYRAMID**.

The tuple (c_0, s_0, y_0, t_0) is a valid signature. Andy provides this to the contract, which checks its validity by running the verification step. It first checks that $y_0 \neq 0$ and then computes $C_0 = g^{t_0} \cdot Z^{y_0}$ and $A_0 = g^{s_0} \cdot \text{PUSIGN}_{\text{Signy}}^{-y_0 \cdot c_0}$. The signature is valid if $c_0 = H(Z||A_0||C_0||\text{ADDR}_{Andy})$. Note that Andy is the only member who can receive this money since his address is embedded into c_0.

9) The contract checks if the sum of the received moneys in this round does not exceed the sum of sent money. If it does, then Signy is dishonest. All transfers in the current round will be ignored and Signy will be punished by increasing her SI to ensure she will never be the signer again. If the check passes, the money will be distributed as follows: Bernie will receive an unlocked MR with a value of $\mu \cdot (1 - \pi)$ and Andy will receive a locked MR with a value of $\mu \cdot \pi$.

This concludes our protocol.

Deposits and Rewards. In practice, to ensure that taking the role of Signy is rewarded, we can pay a fixed small portion of each transfer as a fee to Signy. Similarly, we can require Signy to put down a deposit which will be confiscated if the contract detects her cheating or lack of participation by Signy.

VI. Security Analysis

In this section, we prove the desired security properties of our **PYRAMID** protocol.

Theorem 1 (Valid Blind Signatures)**.** *In **PYRAMID**, Signy cannot provide invalid signatures.*

Proof. Every signature request is recorded in the smart contract and every signature provided by Signy is verified by the protocol contract.

Theorem 2 (Honest Signer)**.** *A rational Signy will follow the protocol honestly.*

Proof. Since she cannot provide invalid signatures as per Theorem 1, a dishonest Signy can either (i) refuse to provide blind signatures for some requests in either of the two mixing workflows, or (ii) forge valid blind signatures that do not correspond to a request recorded in the smart contract. Our contract has deadlines for each step and identifies and punishes (i) as cheating. Since there is no financial benefit in (i), a rational Signy will always provide valid signatures. Attack (ii) is also identified at the last step of every workflow by the contract. If the contract receives more valid unblinded signatures than the number of requests/transfers in the current round, then Signy has forged extra valid signatures. This will invalidate all the referrals and transfers in the current round, which is mildly inconvenient to the other members, but they can retry in next round. Signy on the other hand will be penalized by having her deposit taken away and losing the ability to ever be the signer in the future. Thus, a rational Signy will not attempt this.

Theorem 3 (Signature Reuse Prevention)**.** *A member account cannot reuse the signature in the money flow protocol of a given round to generate a valid signature for a later round.*

Proof. This is a direct corollary of the fact that round numbers are included as public information in the blind signatures.

Theorem 4 (Correctness of Member Referral)**.** *Only members can refer other members. If Andy is a member who refers Bernie, he will become Bernie's parent unless he chooses not to.*

Proof. Only members can request referrals and have them signed by Signy. As per Theorem 2, Signy will not sign blind messages from non-members as this will cause her to be caught for cheating. Since blind signatures are unforgeable, no one else can create them. In Step 2 of our Member Referral workflow, Andy chooses to include $\text{ADDR}_{\text{Bernie}}$ in his c_0. He will do this only if he has access to the private send key corresponding to the public send key in $\text{ADDR}_{\text{Bernie}}$. This by definition means Andy is Bernie's parent. A rational Andy would not agree to add a member Bernie without having control over the private send key. However, there is nothing in the protocol to prevent this. If they so choose, a member can refer another member without becoming their parent. The only side-effect is that the referrer misses out on their shares of the revenue generated by the referee.

Theorem 5 (Correctness of Money Flow)**.** *Each member Bernie can send money only to their parent Andy and can only unlock a locked MR by sending a portion π of it to the parent.*

Proof. Since blind signatures are unforgeable, they can only be created by Signy. As shown in Theorem 2, Signy only signs values c that are sent to her over the smart contract in Step 6. However, in this step, the contract first verifies the consent of Andy by checking that c is signed by the private send key corresponding to Bernie's public send key. Recall that this private key is only known to Andy (Theorem 4). Thus, Bernie can transfer an MR only with Andy's consent, which will be given only if Andy's address is put as the recipient. The contract automatically pays a portion π to Andy (locked) and the rest to Bernie (unlocked) in Step 9.

Theorem 6 (Unlinkability)**.** *If Andy is Bernie's parent, it is infeasible for an adversary, who may control any number of nodes or members other than Andy and Bernie, potentially even including Signy, to identify this relationship. More specifically, in each round of referral, the adversary is unable to distinguish which referrer referred which referee. However, the identities of the sets of referrers and referees of the current round are known. Similarly, in each round, the adversary is unable to distinguish which recipient is receiving money from which sender.*

Proof. This is directly based on the blindness property of our blind signatures. If an adversary can connect Bernie's identity to Andy's, they can distinguish which messages were blind-signed before/after the messages corresponding to the referral/transfer between Andy and Bernie. See [38], [41], [42] for more discussion on the blindness property.

VII. Implementation and Performance

Implementation. We implemented our **PYRAMID** protocol as an Ethereum smart contract in Solidity. The implementation is accessible at [43]. Our code is open-source, free and donated to the public domain with no copyright.

Automated Security Analysis. To gain further confidence that our implementation is secure and does not contain known vulnerabilities, we ran it through two state-of-the-art automated static security analyzers: Slither and Mythril. Slither [44] is an open-source project with over five thousand stars on GitHub. It analyzes Solidity code and reports on a wide range of common vulnerabilities that have been identified by the community. Mythril [45] is a security analysis tool provided by Consensys, a leading smart contract auditing firm. Similar to Slither, Mythril also examines the Solidity code and identifies various types of vulnerabilities. Neither Slither nor Mythril reported any vulnerabilities in our implementation.

Gas Analysis. On Ethereum, deploying smart contracts and interacting with them requires paying gas fees. This is the mechanism that ensures safety against DoS attacks. Since every member has to pay for gas, it is important for blockchain-based protocols to be efficient and use as little gas as possible. The gas costs of our protocol can be divided into deployment costs, i.e. one-time costs for publishing contracts on the blockchain, and workflow costs.

Contract	Deployment Gas Cost (Units of Gas)	Deployment Gas Cost (USD)
Member Account	2,391,290	87
Main Protocol	3,271,747	120
Referral Mixer	1,094,354	40
Money Mixer	1,420,075	52

TABLE I
DEPLOYMENT GAS COST OF THE CONTRACTS IN **PYRAMID**.

Workflow	Function	Gas Cost (Units of Gas)	Gas Cost (USD)
Referral	sendReferRequest	74,427	2.7
	signReferRequest	71,737	2.6
	onBoard	174,398	6.4
	referValidityCheck	55033	2.0
Money Flow	sendTransaction	106,472	3.9
	signTransaction	73,233	2.7
	receiveTransaction	117,275	4.3
	moneyValidityCheck	54,956	2.0

TABLE II
WORKFLOW GAS COSTS IN OUR IMPLEMENTATION OF **PYRAMID**.

The deployment gas cost for each contract in our architecture is shown in Table I. USD values are approximated based on typical gas prices and exchange rates on June 20th, 2024. The main contract takes almost 120 USD to deploy, but this has to be done only once. Each member account requires almost 87 USD in gas fees to deploy. Note that each member creates their account only once, too. We then have one instance of the mixers for each round, so the costs of deploying them can be divided among all members taking part in that round.

The workflow gas cost is incurred when a member interacts with the protocol in the two main workflows: Referral and Money Flow. The gas cost for each step in these workflows is shown in Table II. These costs are significantly smaller.

VIII. Conclusion

In this work, we presented **PYRAMID**, a secure, decentralized and trustless protocol that implements multi-level marketing using smart contracts and ensures privacy and unlinkability of participants to their referrers. We achieved this by relying on two mixing stages, one for referrals and one for money transfers, which both use (partially) blind signatures. We proved that our approach satisfies the desired security and privacy properties and implemented it as a free and open-source Solidity smart contract for Ethereum. Finally, we showed that our contract is gas-efficient and affordable as illustrated in Table II.

Acknowledgments and Notes. The research was partially supported by the Hong Kong Research Grants Council ECS Project Number 26208122. G.K. Conrado was supported by the Hong Kong PhD Fellowship Scheme (HKPFS). Authors are ordered alphabetically.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[2] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, pp. 1–32, 2014.

[3] H. S. Galal and A. M. Youssef, "Verifiable sealed-bid auction on the Ethereum blockchain," in *FC*, 2019, pp. 265–278.

[4] F. Sabry, W. Labda, A. Erbad, H. Al Jawaheri, and Q. Malluhi, "Anonymity and privacy in bitcoin escrow trades," in *PES*, 2019.

[5] A. K. Goharshady, "Irrationality, extortion, or trusted third-parties: Why it is impossible to buy and sell physical goods securely on the blockchain," in *Blockchain*, 2021, pp. 73–81.

[6] A. K. Goharshady, A. Behrouz, and K. Chatteriee, "Secure credit reporting on the blockchain," in *Blockchain*, 2018, pp. 1343–1348.

[7] K. Chatterjee, A. K. Goharshady, and A. Pourdamghani, "Probabilistic smart contracts: Secure randomness on the blockchain," in *ICBC*, 2019.

[8] K. Chatterjee, A. K. Goharshady, and Y. Velner, "Quantitative analysis of smart contracts," in *ESOP*, 2018, pp. 739–767.

[9] K. Chatterjee, A. K. Goharshady, R. Ibsen-Jensen, and Y. Velner, "Ergodic mean-payoff games for the analysis of attacks in crypto-currencies," in *CONCUR*, 2018, pp. 11:1–11:17.

[10] K. Chatterjee, A. K. Goharshady, and A. Pourdamghani, "Hybrid mining: exploiting blockchain's computational power for distributed problem solving," in *SAC*, 2019, pp. 374–381.

[11] R. Muth and F. Tschorsch, "Tornado vote: Anonymous blockchain-based voting," in *ICBC*, 2023, pp. 1–9.

[12] J. Ballweg, Z. Cai, and A. K. Goharshady, "Purelottery: Fair leader election without decentralized random number generation," in *Blockchain*, 2023, pp. 273–280.

[13] Z. Cai and A. K. Goharshady, "Trustless and bias-resistant game-theoretic distributed randomness," in *ICBC*, 2023, pp. 1–3.

[14] ——, "Game-theoretic randomness for proof-of-stake," in *MARBLE*, 2023, pp. 28–47.

[15] P. Schindler, A. Judmayer, N. Stifter, and E. R. Weippl, "Hydrand: Efficient continuous distributed randomness," in *SP*, 2020, pp. 73–89.

[16] P. Fatemi and A. K. Goharshady, "Secure and decentralized generation of secret random numbers on the blockchain," in *BCCA*, 2023, pp. 511–517.

[17] T. Barakbayeva, Z. Cai, and A. K. Goharshady, "SRNG: an efficient decentralized approach for secret random number generation," in *ICBC*, 2024.

[18] V. P. Abidha, T. Barakbayeva, Z. Cai, and A. K. Goharshady, "Gas-efficient decentralized random beacons," in *ICBC*, 2024.

[19] B. Sriman and S. G. Kumar, "Decentralized finance (DeFi): the future of finance and defi application for ethereum blockchain based finance market," in *ACCAI*, 2022, pp. 1–9.

[20] J. M. Taylor, "The case (for and) against multi-level marketing," *Consumer Awareness Institute*, vol. 1, no. 1, pp. 7–1, 2011.

[21] S. Farokhnia and A. K. Goharshady, "Alleviating high gas costs by secure and trustless off-chain execution of smart contracts," in *SAC*, 2023, pp. 258–261.

[22] ——, "Reducing the gas usage of ethereum smart contracts without a sidechain," in *ICBC*, 2023, pp. 1–3.

[23] Z. Cai, S. Farokhnia, A. K. Goharshady, and S. Hitarth, "Asparagus: Automated synthesis of parametric gas upper-bounds for smart contracts," *Proc. ACM Program. Lang.*, vol. 7, no. OOPSLA2, pp. 882–911, 2023.

[24] D. Chaum, "Blind signatures for untraceable payments," in *CRYPTO*, 1983.

[25] E.-O. Blass and F. Kerschbaum, "Strain: A secure auction for blockchains," in *ESORICS*, ser. Lecture Notes in Computer Science, vol. 11098. Cham: Springer, 2018, pp. 87–110.

[26] H. S. Galal and A. M. Youssef, "Succinctly verifiable sealed-bid auction smart contract," in *DPM*, 2018.

[27] Q. Yue, C. Zhong, and H. Lei, "Quantum sealed-bid auction protocol with post-confirmation based on blind signature," *Quantum Information Processing*, vol. 23, no. 70, 2024.

[28] L. Harn, C. Hsu, Z. Xia, and Z. Li, "Multiple blind signature for e-voting and e-cash," *The Computer Journal*, 2023.

[29] M. Kumar, C. P. Katti, and P. C. Saxena, "A secure anonymous e-voting system using identity-based blind signature scheme," in *ICISS*, 2017.

[30] J. C. P. Carcia, A. Benslimane, and S. Boutalbi, in *GLOBECOM*.

[31] L. López-García, L. J. Dominguez Perez, and F. Rodríguez-Henríquez, "A pairing-based blind signature e-voting scheme," 2014.

[32] A. Goharshady and Z. Lin, "Blind vote: Economical and secret blockchain-based voting," in *IEEE Blockchain*, 2024.

[33] T. Ruffing, P. Moreno-Sanchez, and A. Kate, "Coinshuffle: Practical decentralized coin mixing for bitcoin," in *ESORICS*, 2014.

[34] L. Valenta and B. Rowan, "Blindcoin: Blinded, accountable mixes for bitcoin," in *FC*, 2015, pp. 112–126.

[35] N. Lu, Y. Chang, W. Shi, and K.-K. R. Choo, "Coinlayering: An efficient coin mixing scheme for large scale bitcoin transactions," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 3, pp. 1974–1987, 2022.

[36] J. Du, Z. Ge, Y. Long, Z. Liu, S. Sun, X. Xu, and D. Gu, "Mixct: Mixing confidential transactions from homomorphic commitment," in *ESORICS*, 2022.

[37] Z. Bao, W. Shi, S. Kumari, Z.-y. Kong, and C.-M. Chen, "Lockmix: a secure and privacy-preserving mix service for bitcoin anonymity," *International Journal of Information Security*, 2020.

[38] S. Tessaro and C. Zhu, "Short pairing-free blind signatures with exponential security," in *EUROCRYPT*, 2022.

[39] F. Benhamouda, T. Lepoint, J. Loss, M. Orrù, and M. Raykova, "On the (in)security of ROS," in *EUROCRYPT*, 2021.

[40] S. J. De and A. K. Pal, "Auctions with rational adversary," in *Information Systems Security*, 2013.

[41] D. Schröder and D. Unruh, "Security of blind signatures revisited," in *PKC*, 2012.

[42] M. Abe and T. Okamoto, "Provably secure partially blind signatures," in *CRYPTO*, 2000.

[43] K. N. L. Nguyen, "PYRAMID: a protocol for private and trustless multi level marketing on the blockchain," https://github.com/longnguyen1802/PYRAMID-A-Protocol-for-Private-and-Trustless-Multi-level-Marketing-on-the-Blockchain, 2024.

[44] Trail of Bits, "Slither: Static analyzer for solidity and vyper," https://github.com/crytic/slither, 2023.

[45] Consensys, "Security analysis tool for evm bytecode. supports smart contracts built for ethereum, hedera, quorum, vechain, rootstock, tron and other evm-compatible," https://github.com/Consensys/mythril, 2024.