**Decentralized Secure Collaborations (DeSC): Whitepaper**

**Zubayr Khalid**

**Khabbab Zakaria**

**Arun Teja**

**Pritam Paul**

**Abstract:**

*This paper presents the DeSC (Decentralized Secure Collaboration) framework for managing decentralized collaborations. The system facilitates interactions between initiators, investors, and acceptors through a series of smart contracts. Initiators can secure investments from investors via the* ***Investor Smart Contract****, which manages funding agreements and fund distribution through multi-signature wallets. Once funding is in place, collaborations are formalized using the* ***Initiator Smart Contract*** *and the* ***Acceptor Smart Contract****.*

*The core of the framework is the* ***atomicity*** *concept, where collaborations are broken into discrete, verifiable levels. Each level is scrutinized by a randomly selected subset of the* ***Moderator Universe****, forming a* ***Moderator Set*** *that is tasked with validating the collaboration’s progress. Moderators participate in a voting process, including an optional* ***Motion for No Confidence (MNC)****, to address conflicts and ensure fairness. An MNC can be called if a significant portion of moderators detects discrepancies, with the outcome affecting the moderators'* ***Soulbound Token (SBT)*** *credits.*

*Privacy is maintained through encrypted off-chain communication channels, ensuring that only relevant parties access collaboration details. This approach integrates decentralization, security, and scalability, providing a robust solution for managing complex collaborations while reducing risks associated with centralization and fraud.*

**Section 1: Introduction**

Blockchain technology has been implemented in various domains since its inception. It first gained popularity as the underlying technology for the cryptocurrency Bitcoin, providing a secure and transparent way to record transactions without the need for a central authority. Beyond finance, blockchain has also found applications in healthcare [4’,5’], where it is used to securely store and share patient data, ensuring data privacy and security. In the security domain, blockchain technology is being used for secure authentication and identity management [6’]. It has also found a place in the gaming industry, where it is being used to create decentralized gaming platforms [7’], enabling players to own and trade in-game assets securely. Furthermore, blockchain is being used in supply chain management to provide an immutable and transparent record of a product's journey from producer to consumer [8’-11’].

Based on the examples it should be pretty clear that not necessarily all blockchain implementations should use cryptocurrency or a crypto based token. For instance, blockchain technology has various implementations in use cases involving a distributed storage or distributed database and also use cases where a decentralized medium is maintained to reach certain objectives.

In any blockchain application where one or more cryptocurrency and/ or token(s) are involved, it is most certain that a sender will send or store some amount of a currency (fiat or crypto) in a crypto platform. In many of such scenario, a user will just store some currency in his private crypto wallet (e.g. Metamask) or his account in a crypto trading platform (e.g. Binance). Providing user does not share his secret passphrase or password, this type of transactions are generally safe. Another type of transactions can be where a sender will send some asset to gain some different resource (not necessarily crypto asset) from the receiver. In daily life scenario, we can relate it with buying a pizza box from a pizza store, where user sends some currency and receives pizza from the store. Considering the fact that crypto trading happens online without the authority of a centralized party, many things in such transactions can go wrong. For instance, the pizza store can deny giving pizza to the buyer, or maybe can give less amount of pizza in return. In this scenario, the pizza shop is the adversary. In a separate scenario, the buyer can falsely claim sending more currency than he actually did, making him the adversary.

In this paper, we have worked with a secure collaboration, thus Decentralized Secure Collaboration or DeSC. With DeSC backed by the decentralization of Blockchain, we enable a full secure and decentralized collaboration of multiple parties. The application of this paper can be myriads of industries- online collaboration (specially via Social Media), online marketing and transactions, preventing many online scams, etc. are just a few examples. The total possible market is in multiple billions, if not more, which is discussed later in the study. Although there are a few competitors who have been already working on similar domains (discussed later), nobody has fully explored the domain of secure collaboration. Moreover, to our knowledge, none of them is fully decentralized. Thus, they are prone to failure, corruption, and other several drawbacks.

**Section 2: A Study on Scams involved in Multi-Party Financial Transactions**

The use of DeSC presents an efficient solution for countering fraud and scams, particularly in scenarios involving financial transactions among multiple parties. In this section, our focus is to examine the frequency and occurrence of recent fraudulent activities within various sectors, specifically those involving financial transactions among two or more parties.

Firstly, we'll delve into scams prevalent within the sphere of social media. Undoubtedly, social media platforms have become an integral part of modern life, enveloping our daily experiences. Giants such as Facebook, Instagram, WhatsApp, Twitter, Snapchat, and YouTube dominate discussions, garner high viewership, and constitute subjects of extensive study in today's world. Financial transactions within these platforms have become increasingly common.

Consider Instagram, which initially emerged as a benign platform for sharing photos and videos. However, since its acquisition by Facebook, now Meta, it swiftly evolved into a hub fostering business interactions among its users. Presently, numerous Instagram influencers boast substantial followings, serving as an unparalleled conduit to reach expansive audiences. From renowned Hollywood actors engaging in 'paid partnerships' with established brands to lesser-known influencers endorsing local products, the spectrum is vast. Omnicoreagency.com reports a staggering 500,000 active influencers on Instagram, with approximately 37% of Instagram users engaging with these influencers, totaling around 74 million [1 ].

Let's delve deeper into the realm of frauds within this domain.

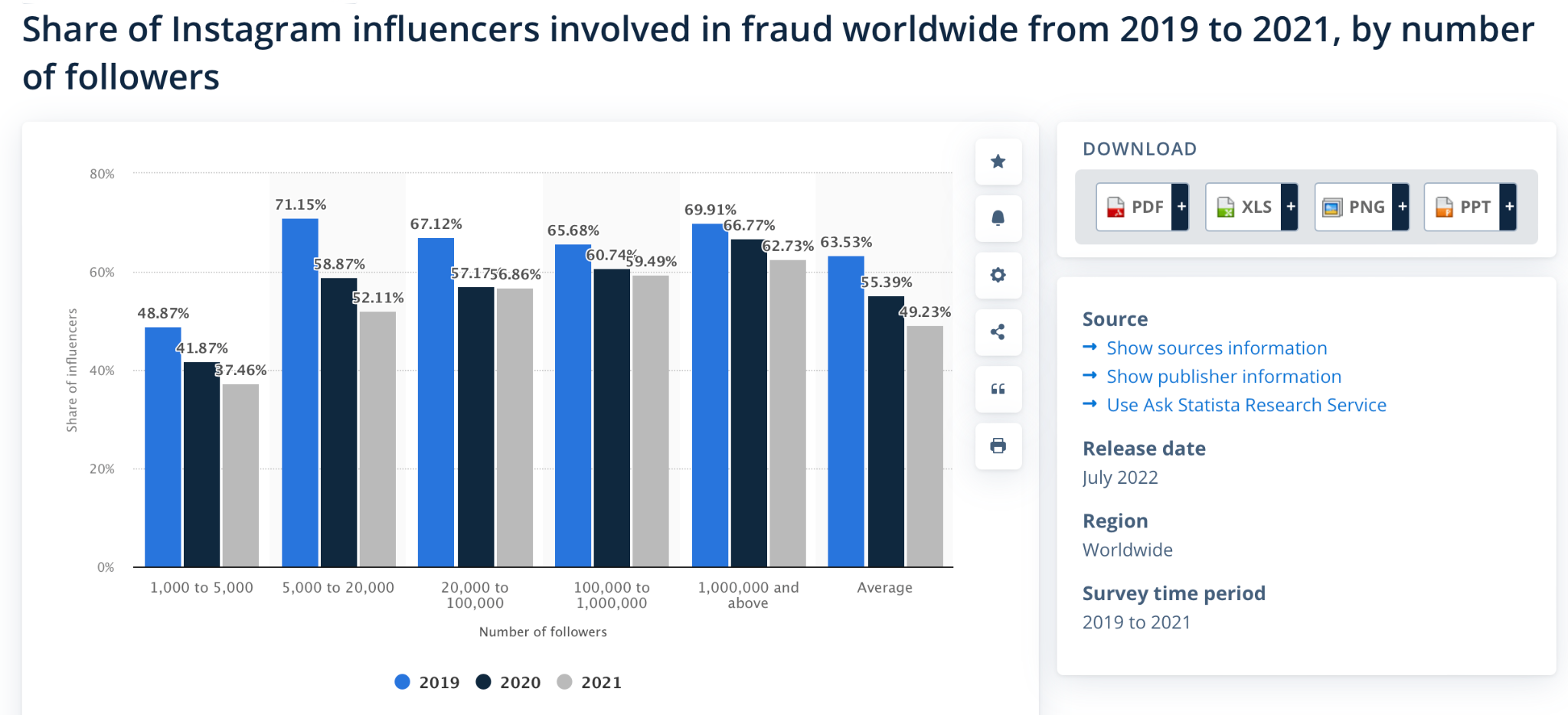


Image courtesy: Statista [2]

The provided image depicts the percentage of Instagram influencers engaged in fraudulent activities, displayed through bar charts spanning three years: 2019, 2020, and 2021. These statistics segment influencers based on their follower counts. Notably, the data reveals an alarming trend, indicating that, on average, 67% of influencers boasting over 1 million followers have been involved in fraudulent activities throughout these three years.

It's crucial to note that many instances of fraud within this sphere go unreported for various reasons. Factors such as blackmail, inadequate customer support, and users' lack of knowledge regarding self-defense mechanisms contribute to the underreporting of fraudulent activities.

Most common types of frauds in social media are:

1. Phishing Scams,
2. Fake Investments,
3. Fake Collaboration Requests,
4. Selling Followers and Likes,
5. Giveaways,
6. Non-Existent Job Offers,
7. Imposter Brand Account.

The enormity of social media frauds is evident from the substantial figures reported by major platforms. Facebook, for instance, consistently removes billions of fake accounts annually, with Meta's reports indicating 1.6 billion and 1.4 billion fake account removals in Q1 and Q2 of 2022, respectively [3]. These fraudulent accounts, comprising 5% of Facebook's global monthly active users in Q2 2022, often engage in criminal activities, including financial scams.

Similarly, LinkedIn, catering largely to professional domains, encountered and actively tackled millions of fake accounts in 2021. The platform prevented 11.9 million fake account creations, proactively restricted 4.4 million accounts, and took action against 127,000 reported fake accounts. Moreover, it removed 70.8 million accounts engaged in spamming or scamming, along with 179,000 reported by members [4, 5]. Most of these deceitful accounts peddle non-existent job opportunities, impersonate representatives of prominent companies, or offer services for payment, intensifying the risk of impersonation and fraudulent activities akin to those observed on Instagram [6].

The Federal Trade Commission's data spotlight underscores the financial impact, with more than 95,000 individuals in the USA reporting losses of approximately $770 million due to social media-initiated frauds in 2021. This accounted for a staggering 25% of all reported fraud losses in that year, signifying an 18-fold increase from figures reported in 2017 [7]. Moreover, CBS News reports that advertisers lost around 15% of their spending on influencer marketing—equating to $1.3 billion in 2020 and projected to reach nearly $2 billion in 2022—due to fraudulent practices [8].

The surge in popularity of cryptocurrencies (cryptos) and Non-Fungible Tokens (NFTs) has led to an exponential rise in associated scams. These scams often capitalize on collaborations with influencers boasting massive followings across various social media platforms.

“*The amount of total counterfeiting globally has reached to 1.2 trillion USD in 2017 and is bound to reach 1.82 trillion USD by the year 2020 which includes counterfeiting of all equipment/ products from defense equipment to counterfeiting of watches.*” The 2018 Global Brand Counterfeiting Report [9] estimates that the losses suffered due to online counterfeiting globally has amounted to 323 billion USD in the year 2017. According to the analysis made in the report, losses incurred by Luxury Brands because of sale of counterfeiting through the internet accounted for 30.3 billion USD.

Of course, another big market is e-commerce. According to statista, Retail ecommerce sales are set to reach $5.5 trillion in 2022 [10]. There are about 8 billion people worldwide of which about 2.14 billion people shop online [11].

### According to Payments Dive, eCommerce fraud caused losses of $20 billion in 2021 [12].



Image courtesy: statista [13]

According to statista, $41 billion globally was lost on e-commerce payment fraud in 2022 [13].

In ecommerce, when a chargeback happens, it becomes cumbersome for the retailer themselves as they have to prove that the product was successfully delivered. Many customers take unfair advantage of this situation. They apply for chargeback even after receiving the product. This is also called friendly fraud. According to Verifi, businesses lose $308 for a $100 chargeback [14]. According to LexisNexis [15], this amount is almost $400. According to Cybersource [16], this is the top fraud affecting the merchants in all domains. According to Chargeback911 [17], the number of internet scam complaints is growing rapidly as shown in the figure below:

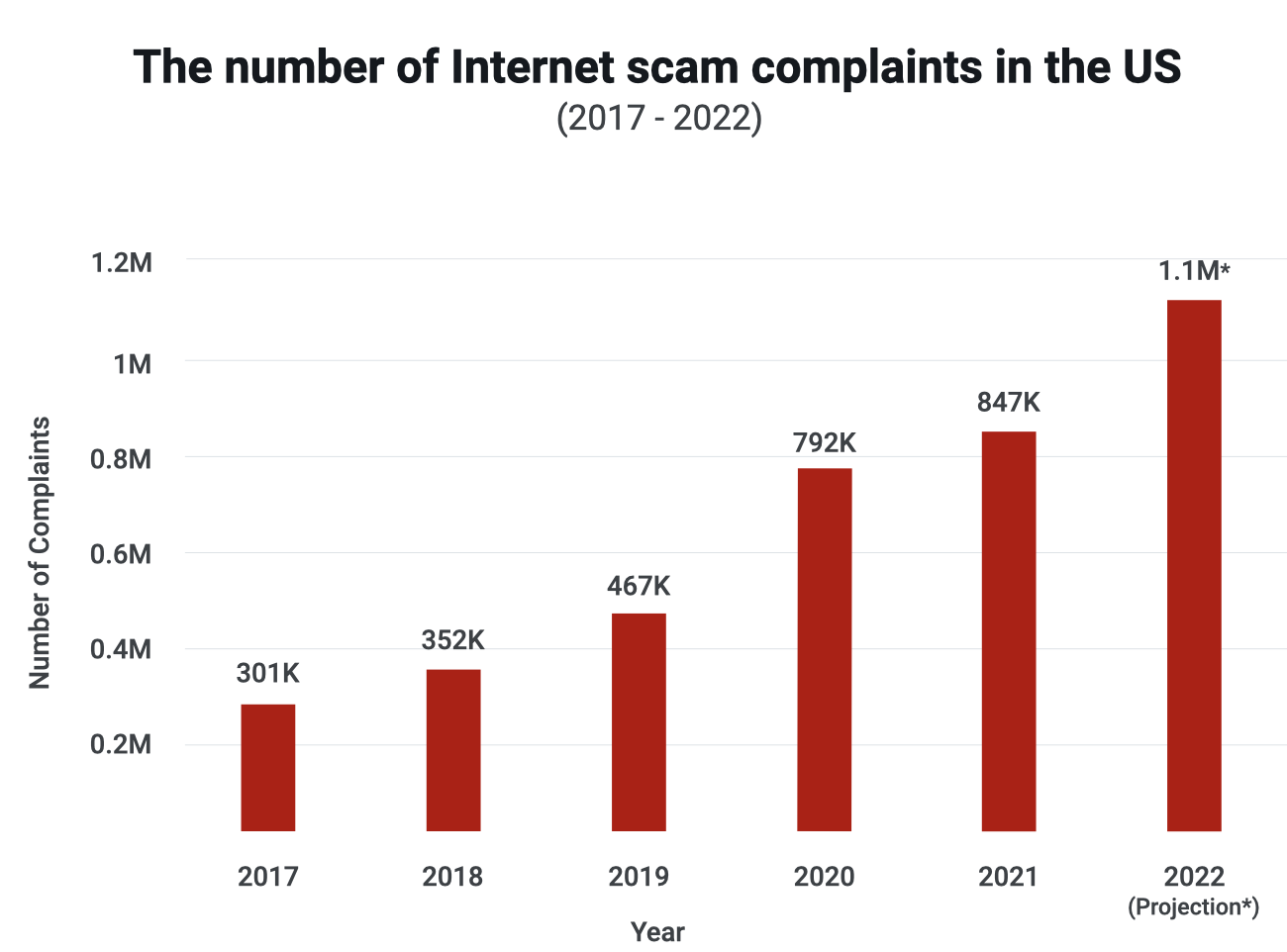


Image Courtesy: Chargeback911 [18]

According to Chargeback911, by 2023 6 out of 10 chargebacks in North America will be fraudulent [18]. According to Forbes [19],, chargebacks can account for between 40% and 80% of all eCommerce fraud losses. Mastercard [20] states that 80% of chargebacks are fraud-related. This includes both third-party (“criminal”) fraud, as well as first-party (“friendly”) fraud.

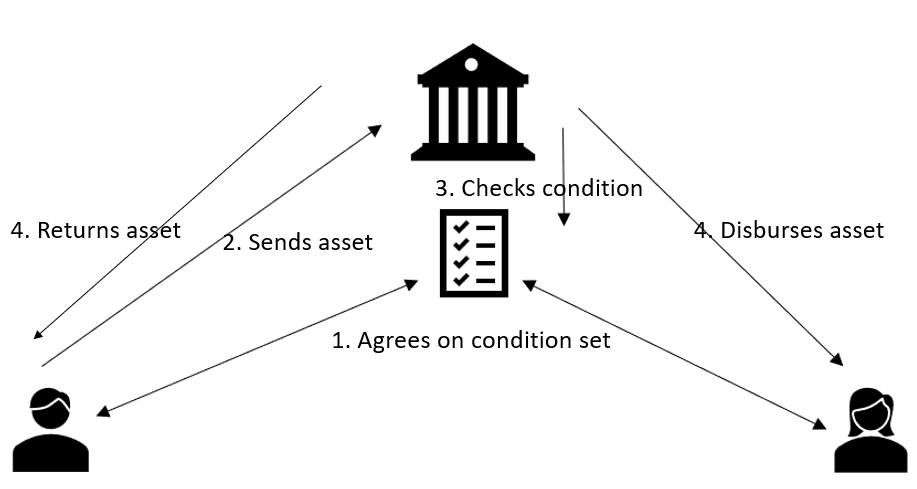
However, we do need chargeback as the ‘last resort’ of the customers against credit card frauds.

A new study from [Juniper Research](https://www.juniperresearch.com/home) [21] has found that the total cost of eCommerce fraud to merchants will exceed $48 billion globally in 2023, from just over $41 billion in 2022. According to the Nilson Report [22], by 2030, total payment card volume is expected to top $79 billion. Of that amount, an estimated $49.32 billion will be lost to criminal fraud.

Finally, fraud costs the global economy $5.127 trillion each year [23].

**Section 3: EXISTING APPLICATIONS**

Any business deal between 2 untrusted parties can be moderated by a third party that the other parties mutually trust. This can be defined by Escrow protocol. Escrow is a contractual agreement between two parties in which a trusted third party receives and holds the asset on behalf of the sender. The third party is usually called the escrow agent. The agent monitors the deal and either disburses the asset to the receiving party (in case the deal is complete) or returns the asset to the sender party (in case the deal is not fulfilled). The agent must follow some specific conditions set by the main 2 parties. The conditions are set prior to creating an escrow account. Not necessarily all the conditions must be satisfied in order for the deal to go through. There can be scenarios where if a majority of the conditions (say 2 out of 3) are satisfied then the agent can decide that the deal is complete. When the 2 main parties decide to create an escrow account, they agree on a set of conditions (can be provided by either of the parties), and the sender party deposits the asset to the account. Both parties agree on an escrow agent (can be a bank or a single entity). The agent remains responsible for the asset until a specific time period. If the conditions are satisfied within the agreed time period then the agent disburses the asset to the receiver party. If some of the conditions fail to satisfy or if the time period ends before the objective is completed then the agent returns the asset to the sender party.



Up next we will discuss some common examples that require the use of escrow protocol.

*A. Real Estate Business*

A very popular use case application for escrow protocols is real estate transactions. In the real estate business, a trusted party is hired to hold all documents and assets on behalf of the buyers and sellers. The buyers are usually individuals or a collective entity and the sellers are property providers. The third parties are generally law firms or escrow companies. In the real estate business, when there is an agreement between both parties, there is often a down payment involved. The down payment is usually a part of the total cost of the property which the buyer must provide to the seller as an initial investment. The parties agree on a set of terms such as the number of installments for the rest of the payment, and legal documents from both parties. If an escrow protocol is involved, then this investment amount along with the conditions and documents are held by the third-party escrow agent. In case the buyer backs out or the agent finds any loophole in the documents provided by the buyer, then the initial investment amount is sent to the seller as compensation. On the other hand, if the seller backs out or becomes unable to provide the property within the agreed time period then the amount is returned to the buyer. During the initial agreement, the seller can also add some compensation fees that the buyer gets in case the seller backs out. In case multiple installments are involved, the buyer can send these payments to the agent as well. The agent, keeping a separate account for the next payments, sends only the down payment to the seller and the rest of the amount to the buyer in case the buyer backs out. In case the buyer sends the whole amount and the seller provides the property on time, the agent sends the whole amount (from both of the accounts) to the seller.

*B. Import-Export Business*

When 2 untrusted parties are involved in trading it is often a custom to have a third-party moderator. Either of the main parties can be a private company or the Government. An escrow protocol is needed in this use case because it can eliminate the possibility of the buyer having a damaged asset (or not having the asset at all) after advanced payment, as well as the risk of the seller not having the full payment after the asset is shipped. Once both parties settle on the trading cost, volume, and time of delivery, the buyer party sends the amount to the escrow agent. After receiving the asset from the seller party, the buyer party can check the condition of the asset and send a ‘green light’ to the agent. The agent, upon receiving the validation from the buyer, sends the amount to the seller party.

The escrow protocol is also applicable to e-commerce applications and other online marketplaces. In this scenario, after ordering a specific asset from an e-commerce application, the buyer sends the amount along with other necessary documents to the escrow agent. Upon delivery, the buyer uses a predetermined amount of time to inspect and accept the asset. Once accepted, the funds are released by the escrow agent to the e-commerce application. In case of a dispute (damaged asset or not receiving the asset within a specific time period), the agent can investigate the matter. Since the buyer themselves can damage the asset or deny receiving the asset to avoid sending the amount, one solution for the agent to properly resolve the dispute can be to use a supply chain management scheme for the overall shipment of the asset. This solution is very effective but it needs further involvement to implement the supply chain in trading. In many cases, this solution also needs multiple escrow agents to properly validate the shipment. Perhaps a simpler solution can be for the seller to ship the asset to the escrow agent, who in terms, can validate the quality of the asset and finally send it to the buyer.

*C. Crowdfunding*

It is quite common to use escrow protocol in crowdfunding. A crowdfunding campaign is involved when typically a startup initiates a project that requires funding. The main goal is to invite investors to donate funds in exchange for specific privileges such as stakes. The project coordinators need to provide details of the project along with a ‘target amount’ of funds that is required to fuel the project. In different crowdfunding applications, it is a mandatory criterion to mention the target amount and the time period of the campaign. In case the target amount is not reached within the predefined period of time, it is considered that the project does not have the necessary funding to be started. An escrow agent is required in this scenario who can return the existing funds to its actual investors in case the target amount is not reached. The general concept is, the escrow agent will hold all the funds (along with the details of the investors) until the end of the campaign. After the campaign time period is over, if the agent finds out that the target amount is reached, then he can send the whole amount to the project team. Otherwise, the agent will conclude that the campaign has failed to get the necessary amount and will return the funds to the investors.

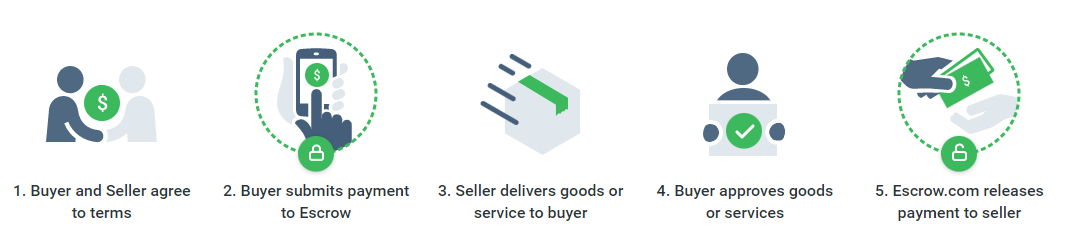
Up next we will discuss some of the most common escrow applications.

*A. Centralized Applications*

**1. Escrow.com:**

It provides a centralized escrow service that allows users to buy and sell assets through a trusted third-party moderator (broker). The application’s three-party transaction gives Brokers the ability to start, manage, and close every sale privately. Escrow brokers provide a secure and neutral third-party service to facilitate transactions between two parties. They ensure that the terms of the agreement are met before releasing the funds or property to the appropriate party. All the 3 parties need to register to escrow.com in order to agree to terms. Once the parties agree to terms, the broker initiates the transactions. Next, the buyer pays the amount to escrow.com. Upon receiving the payment, escrow notifies both the seller and broker. In the next stage, the seller sends the asset and additional documents to the buyer. Escrow.com verifies that the buyer has received the asset correctly and the broker is notified as well. The buyer uses a predefined number of days to inspect the asset and either accept or reject it. In case the asset is accepted by the buyer, the seller and the broker are paid accordingly by escrow.com.

The flow diagram (src: escrow.com) below describes how the protocol works with Escrow.com.



If the buyer rejects the asset, they are required to return it within 10 days. The Seller gets a 5-day period to inspect the returned asset. Based on the Seller's acceptance of the returned item, the Buyer is refunded the original payment less the entire escrow fee and any original shipping fees.

Involved assets include domain names, vehicles, electronic items, merchandise, jewelry, and watches.

Escrow.com provides multiple ways of broker transactions.

*Transparent to buyer:* transaction information such as asset price, seller’s proceeds and contact information can be visible to the buyer, seller, and broker.

*Transparent to seller:* financial terms (asset price, escrow fee, broker commission) are only visible to the seller.

*Transparent to buyer and seller*: financial terms are visible to all parties.

*Confidential:* financial terms are not visible to the buyer and seller. Both the broker commission and escrow fee can be paid by the Buyer, Seller, or split between the Buyer and Seller.

Managing Disputes:

In an event where the Buyer rejects the assets, they will use the seller's contact information to initiate a return. Escrow.com makes sure any complaints from either party are properly communicated, and that the Broker is kept fully informed at all times. In the event of a cancellation (after funds are received) or rejection of merchandise, the Buyer is responsible for the escrow fee regardless of whomever (Buyer, Seller, or Broker) is set up to pay the fee.

**2. EscrowGuardian:**

EscrowGuardian is a type of service that provides a secure, neutral third-party intermediary for financial transactions. It is typically used to facilitate the buying and selling of goods and services, particularly when one or both parties are located in different geographical locations. The escrow service holds onto the funds or assets being exchanged, and releases them to the appropriate party only after certain predetermined conditions have been met. This can help to reduce the risk of fraud and ensure that both parties are satisfied with the outcome of the transaction. EscrowGuardian is often used in real estate transactions, online marketplaces, and other situations where a neutral third-party is needed to hold and disburse funds.

EscrowGuardian holds onto the funds or assets being exchanged by placing them into a secure, segregated account. This account is typically managed by a neutral third-party, such as a bank or trust company, and is used specifically for the purpose of holding funds or assets in escrow. The account is often set up in such a way that the funds or assets cannot be released or transferred without the authorization of all parties involved in the transaction.

*B. Decentralized Applications*

**Smartlink:**

It is a blockchain-based application that executes commercial transactions through a decentralized escrow solution where smart contracts are implemented over the Tezos blockchain. Smartlink allows users to use a set of secure, audited, and pre-defined smart contract templates by utilizing the Smartlink smart contract library (described in the paper as Sscl). However, users can also customize or create new contracts specific to their requirements. The Smart contracts allow multiple stakeholders. Besides, they have a multi-step agreement protocol that allows involved parties to create several payment milestones. The involved parties in an escrow agree on a smart contract (from Sscl or create a new smart contract). The smart contract allocates an inspection period before the closure of a deal to ensure that the involved parties can inspect and confirm if the smart contract accommodates the prerequisites. After both parties agree with the terms defined in the smart contract the buyer provides some stipulated amount of shares (tez) that gets stored in the smart contract wallet. Next, the seller has to deliver the asset within a specific period of time. If both parties verify that all the conditions mentioned in the smart contract are satisfied, the escrow smart contract releases the withheld amount to the seller. A Mediator is potentially involved if there is a dispute between the buyer and the seller.

Smartlink provides the native token named SMAK. However, through the FA 1.2 standard, one can implement user-defined tokens as well. Smartlink also utilizes the FA2 standard to transfer NFTs.

*Smartlink Functionalities:*

A. Transferring Digital Asset:

The buyer and seller must connect their Tezos-compatible wallets to the Smartlink escrow services through the Smartlink API. There are 3 separate smart contracts involved: SMAK smart contract, Payment smart contract, and Escrow smart contract.

The whole procedure is as follows:

If the stakeholders agree and validate the transaction, and sign the transaction with their wallet, the Payment smart contract will create the payment and permit. Next, the Smartlink API will verify the payment data. The stakeholders will be notified regarding created payment in a separate email. If the created payment complies with the preset requirements then the involved parties will form a consensus.

The escrow can be made through the escrow smart contract if both stakeholders comply after they view and validate a transaction. To create an escrow, the parties will interact with the escrow smart contract. The permit is verified by connecting with the Payment smart contract and the escrow validity is confirmed by the Smartlink API. The stakeholders are then notified in separate emails and they become legally bound by the escrow.

B. Transferring Physical Asset:

The main difference between transferring physical assets and digital assets is that transferring physical assets potentially needs using and maintaining off-chain data. Furthermore, the involved parties may require end-to-end tracking of the physical asset.

The Smartlink Escrow Service utilizes several on-chain and off-chain mechanisms. The mechanisms verify the identity of the involved parties and allow end-to-end tracking of the physical assets. The verification of the identity of the involved parties is done by a built-in KYC that holds potential malicious actors responsible for any harmful behavior.

For the exchange of physical assets, there is always a shipping entity responsible for delivering the asset to the buyer. The shipping entity may or may not be the seller, and may rely on

subsidiary shipping services depending on circumstances.

The problem especially when multiple intermediate parties are involved is how to authenticate them, which may be dynamically changed during the shipping process.

Another challenge is collecting the accord from the final users for them to participate, and the proof that each intermediary complied with its corresponding function. It is to ensure the chain of custody of the goods being shipped, which is required in order to establish liabilities of the different intermediaries in case of disputes. The solution provided by Smartlink is to link up with identity providers and/or KYC providers to verify authentication, especially for the involved intermediate parties.

Interaction with SMAK, Payment, and Escrow smart contract is done on-chain and identity providing, KYC, and storing information on the shipment is done off-chain. Oracle is used to store off-chain states in the blockchain.

Smartlink allows both off-chain and on-chain models of authentication. In on-chain authentication dynamically providing identities from the off-chain world are stored on-chain.

Identity is either provided by a Certification Authority or by the Blockchain and Social Media Account Pair pattern.

In the first process, the subject requests a CA for the digital certificate. The certificate can be validated by any entity which is in possession of the public key of the CA. Once received, the subject forwards the certificate and the signed identity to the Adapter Smart Contract. An Oracle is used to check the validity of the certificate before posting it on-chain in the identifier registry.

In the second process, instead of involving a CA, the subject posts the generated public key along with the signed identity on a social media account and then forwards the link to the Adapter smart contract.

In the process of transferring custody in the exchange of physical assets, after the recipient is successfully authenticated by the Smartlink API, the Smartlink API generates a transaction to invoke the Escrow contract to query for the new custodian. Next, it is validated by the dealer's wallet and issued to the blockchain together with the public key of the recipient and with proof of the integrity of the asset.

In a scenario where the recipient does not accept the proof of integrity, or the asset is not delivered within the predefined time frame, the recipient will reject the custody of the asset and either of the parties can raise a dispute.

For dispute resolution, the system relies on a built-in chat feature dedicated to solving disputes on the platform and the potential intervention of a mediator if the parties can’t agree to the new terms. Besides, the Resolve Disputes Online (RDO) features an AI-enabled mediation module for high-volume dispute management.

*Important Features of Smartlink:*

a) Smartlink provides the native utility token SMAK over Tezos blockchain.

b) Smartlink provides on-chain decentralized governance.

c) Smartlink provides rewards to consumers for each transaction they have processed in SMAK.

*Limitations of the current escrow protocols:*

A. The truthfulness of the centralized protocol is based on the assumption that the centralized service provider is not an adversary.

B. For all the existing protocols the service is limited to buying and selling digital and physical assets. In our work, we will introduce a bigger domain and describe how the DeSC can be useful for those use cases.

C. The existing applications allow only 1 moderator in case there is a dispute, which is pretty centralized even if the main application is decentralized (e.g. Smartlink). In DeSC we will introduce the moderator set where multiple numbers of moderators will act to resolve a dispute.

D. In the existing applications, the moderator is only involved at the end of the operation. We believe, in order to allow moderators to properly resolve a dispute, moderators should be active in multiple stages. Hence we have introduced the idea of dividing the operation into atomic stages, where at the end of each stage there is a verifying stage by the moderators.

E. Based on the methodologies followed by existing applications, it is obvious that the buyer information, seller information, moderator information, and other relevant information such as information on the asset, and its delivery date are public (even if not public to any other entity, but public to all the involved parties). In DeSC, all the user information is private, though they can verify their identity using zero-knowledge proof. Besides, due to dividing the operation into atomic stages, involved parties can include different moderator sets for different stages. Moderators involved in one specific stage can learn the sub-operation involved only in that stage, but not the whole operation.

*Further benefits that we want to provide with DeSC:*

A. Similar to Smartlink, DeSC also requires users to provide KYC during registration. Users need to log in to a centralized portal to access the application interface.

B. DeSC will include the idea of Soul Bound Tokens (SBTs) which will be used to maintain a rating system for all the involved parties.

C. **Inclusion of Defi**: In DeSC, other than the buyer entity, seller entity, and moderator set, a fourth type of entity set is the investor set. For collaborations where there is a transfer of assets (digital or physical), the buyer can request grants from the investor set. A separate smart contract will be responsible for creating the collaboration between the buyer and the investor that will include the total amount of the grant, the deadline to return, and the commission amount.

**Section 4: Decentralized Secure Collaborations (DeSC)**

Decentralized Secure Collaboration or DeSC, as the name suggests, is an application that provides secure collaborations among untrusted parties. While discussing probable collaborations between the buyer and seller we only talk about transferring assets (the features provided by current escrow applications). But we always overlook that there can be other types of use cases that may require untrusted parties to collaborate.

Some of such collaborations are as follows:

a) Freelancing Service: In this scenario, an entity may require a freelancing service from a second party. For example, the authority of a local coffee shop may ask a freelance web designer to create a website for them. Note that there is no transfer of assets, rather completing a certain objective. Though in both cases verification is needed (verification that the asset is transferred properly and the verification that the objective is completed properly).

b) Contract-Based Job: For scenarios where a company hires an employee for a certain job, it is needed to verify that the objective was done by the employee and the agreed payment was done by the company.

c) Influencer Marketing: Since social media has taken a giant leap in recent years, it is quite common among both big and small companies to collaborate with social media influencers to market their product. It is important to use a verifiable collaboration platform in this use case since scamming is becoming very common in this field, especially in the crypto world.

d) Crowdfunding: As discussed earlier, a verifiable secure collaboration platform is needed for crowdfunding campaigns. If we consider dividing the campaign into atomic levels (each level being moderated by moderators) then not only it is possible to verify if the total target amount is reached, but parties can also agree on several other decisions (such as agreeing on increasing/ decreasing the target amount) based on parameters such as campaign initiator’s progress in the project during several stages of the campaign as well as the volatility of the market in different times throughout the campaign.

Definitions:

Since DeSC is more than import-export trading, we can't rely on the concepts of ‘buyer’ and ‘seller’. So in this subsection, we will redefine our entities and we will use these names for the rest of the paper.

a) Initiator: Instead of a buyer we have an initiator. An initiator is an entity that initiates a collaboration.

b) Acceptor: Instead of a seller we have an acceptor. An acceptor is an entity that accepts the collaboration request from an initiator.

c) Moderator Universe: It is the set of all the moderators available in DeSC.

d) Moderator Set: It is the set that will monitor a particular collaboration. The moderator set is a subset of the moderator universe.

e) Investor Set: The investor set includes entities that can provide grants to the initiator for a collaboration.

f) Collaboration: It is the whole campaign between the initiator and the acceptor.

g) Atomicity: It is the property of dividing a collaboration into atomic levels.

In DeSC, an initiator initiates a collaboration. If the initiator needs investments, then it needs to call - **Investor smart contract**. The smart contract will allow the investor to make a deal with one or more investors from the investor set. In case no investment is needed, the initiator can directly call the - **Collaboration smart contract**. When an acceptor accepts the collaboration smart contract, it will allow the initiator and acceptor to make a deal. Next, the initiator and the acceptor jointly can divide the collaboration into atomic levels and assign a moderator set for each atomic division. For this whole process, they need to call - **Moderator smart contract**. Note that there can be repetition of moderator sets for multiple atomic stages, and one single moderator set can be assigned for the whole collaboration. It is recommended to provide atomicity to the collaboration. But the initiator and the acceptor can also settle with a one-stage collaboration in which case only one moderator set will be responsible for verification, and similar to existing applications, the moderator set will verify the collaboration after it's completely done. If atomicity is provided to the collaboration then each atomic stage will be verified by its respective moderator set. Without this verification, the campaign cannot move to the next atomic stage. Further note that the involvement of moderators is spontaneous. It does not depend on the **‘dispute resolution call’** by either of the initiator or the acceptor.

If atomicity is provided and a different moderator set is assigned for different atomic stages, then a specific moderator set will only know about the details of the corresponding atomic stage, but not the whole collaboration. Besides, the initiator, acceptor, or moderators will have their identity kept secret from the other entities. In case one entity needs other entities to verify its identity, it can provide a zero knowledge proof. We will discuss it later in detail.

In the next subsections, we will discuss in order the following topics before going to the overall pipeline.

a) Entity Registration and Entity Rating using SBT (SoulBound Tokens)

b) Investor Set

c) Initiation and Acceptance of Collaboration

d) Moderator Universe, Moderator Set, and Atomicity

e) Dispute Resolution

*A. Entity Registration and Entity Rating using SBT:*

Before going further with entity registrations, we need to understand what Soul Bound Tokens are. The concept of SoulBound tokens (SBTs) was proposed by Vitalik Buterin, Glen Weyl, and Puja Ohlhaver, in May 2022 in the whitepaper, entitled “Decentralized Society: Finding Web3’s Soul”. The difference between SBT and NFT is that SBTs are non-transferable tokens representing a person’s identity. In other words, SBTs define the identity of a user, allowing individuals to verify all of their information. While NFTs can be traded among users (via different trading platforms such as Opensea and Rarible), SBTs can not be traded. Just like it is not possible to trade the Social Security Number or Passport with another person. The concept behind Soulbound Tokens seemingly comes from the game World of Warcraft (WoW). In the game, “soulbound” is a property of an item that prevents it from being traded or mailed to another character. SBTs can of course be updated and one user can hold multiple SBTs. For example, one user can have one type of SBT that represents her medical history while another type of SBT that represents her education history. As time passes, both kinds of SBTs may update (not necessarily though). SBTs can not be just ‘mined’ by users (which is possible for NFTs). Rather, SBTs are allocated by entities called Souls. For example, employee rating SBTs are provided by the corresponding company to its employees. SBTs can also be used in other applications such as representing work history, credit history, professional certifications and also user achievements. They can be tied to a myriad of other traits, features, and personal information such as verifying user identities like name, birthday, political affiliations, charitable giving, criminal record, nationality, religious upbringing, military history, and more.

Up until now, anyone can fill their resume (or brag on social media) with false information based on accomplishments that they actually did not achieve. Currently, there is no proper way to verify the correctness of such information. But with SBTs, the accomplishment provider’s (say degree provider from a University) “Soul” (which may be an administrator/controller of exams from the university) would have to grant the user's “Soul” an SBT of a diploma. In this respect, SBTs can be distributed amongst members of a group or institution as proof of affiliation. In this context, a user’s ‘soul’ actually signifies her private wallet. The validity of an SBT can be verified from the provider's soul. So it is not possible to claim false achievements and credentials. Since SBTs are put in the blockchain, it is also not possible to alter the information.

Now that we have the basic idea of SBTs covered, we can move forward with **user registration, its rating system, and user identity verification with Zero Knowledge Proof** in DeSC.

The entity registration is based on the idea that providing signatures is enough to get the SBT rating as well as details regarding other activity (initiating or accepting a collaboration, moderating a collaboration). Anybody should be able to use the signature to retrieve the user’s public key. Since SBT ratings will be linked with the user’s public key, it will be possible to retrieve a user’s SBT rating if the signature is available. In case of disputes, any entity can request any other entity to provide proof of their identity. In that case, the requested entity must provide zero-knowledge proof - “I have a private key s1, a nonce s2 such that from them I can generate public key w1 and my user address w2, where I provide the w1, w2 as the public parameters”.

Note that the user registration platform will be centralized because the benefits provided by decentralization are not needed here. The user needs to provide certain information such as name, nationality, and identity proof in order to register in the system.

Upon registering, users will receive a public address (that will work as their identity in the DeSC platform), a private key, and a public key. The private key is a random integer number with an upper bound of n (n is a large prime). The public key is generated from the private key. The user address is generated by hashing the private key appended with a nonce value provided by the user. A secure cryptographic hash function will be used to perform this operation. The difference between the user address and the public key is that the user address will be used as the identity of the user in the DeSC platform (similar to user’s wallet address; and will be used to call and interact with all the smart contracts) and the public key will be used to link up with the SBT ratings. However, one thing to note is that both the user address and the public key are created from the private key.

**Public Parameters:** User Address, Public Key

**Private Parameters:** Private Key, Nonce Value

Up next we will discuss in order:

1. Key and User Address Generation.

2. Signature Generation.

3. Linking and Updating SBT Rating.

4. Identity Verification with Zero Knowledge Proof

Note that only point 1 is a part of entity registration. Points 2 and 3 are parts of linking with SBT ratings. Point 4 is part of identity verification during disputes.

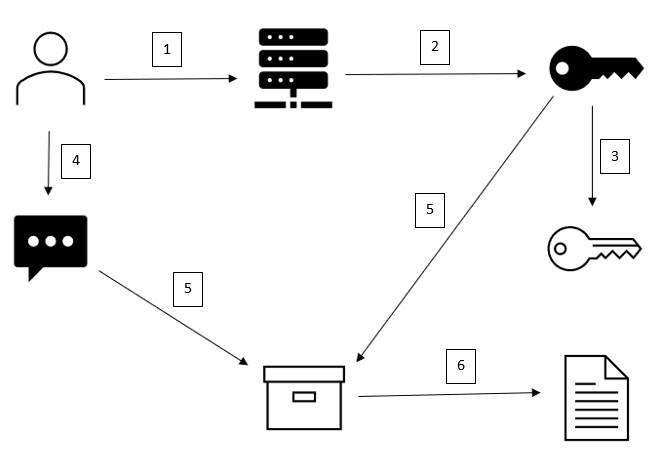


Fig2: 1: User Registration, 2: Private Key is created. 3: Public Key is created from the private key. 4: User provides the Nonce Value. 5: Hash of Private Key and Nonce Value. 6: User Address is created

1. Key and User Address Generation:

We will use ECDSA for key generation.

The private key (privKey) is generated as a random integer in the range [0...n-1].

The public key (pubKey) is a point on the elliptic curve, calculated by the EC point multiplication:

pubKey = privKey \* G, where G is the generator point.

We will use the SHA512 algorithm to generate the user address. For this, the user needs to provide a secret nonce value also in the range [0...n-1].

The user address is generated by the formula:

User Address = SHA512[private key || nonce], where || defines the concatenation operation.

2. Signature Generation:

The algorithm takes a message value (a unique nonce string) along with the user’s private key and generates a signature. We will call the message value as **Collaboration Identifier**.

The algorithm works as follows:

a) Calculates the hash value of the message.

b) Generates securely a random number k in the range [1..n-1].

c) Calculate the random point R on the elliptic curve, where R = k \* G.

Its x-coordinate: r = R.x is noted.

d) Calculate the signature proof: *s* = k−1∗(h+r∗privKey)(mod n)

e) Return the signature {*r*, *s*}.

3. Linking and Updating SBT Rating:

To manage and access Soulbound Token (SBT) ratings linked to a user's public key, we follow these steps:

1. **Linking Ratings:**
   * The public key is used to identify the user.
   * The SBT rating, which represents a particular attribute or score of the user, is associated with this unique public key.
   * The system records this association, ensuring that each public address has a corresponding SBT rating stored securely.
2. **Retrieving Ratings:**
   * When someone wants to find out a user's SBT rating, they use the user’s public address.
   * The system looks up this public address in its records to find the linked SBT rating.
   * The rating is then provided to the requester, reflecting the user’s current status or achievement.

To efficiently link and manage Soulbound Token (SBT) ratings with user public keys, we will use a smart contract named **SBTRegistry**.

The **SBTRegistry** smart contract is a digital ledger deployed on the blockchain that serves as the authoritative source for storing and retrieving SBT ratings. Here’s how it works:

1. **Storing Ratings:**
   * The contract allows an authorized entity to record or update SBT ratings associated with a specific public key.
   * Each user’s public key (a unique identifier derived from their private key) is linked to their corresponding SBT rating in the contract.
2. **Retrieving Ratings:**
   * To find out a user’s SBT rating, anyone can query the smart contract using the user’s public key.
   * The contract provides the stored rating associated with that public key, ensuring transparency and easy access.

To ensure that only valid and authorized updates to SBT ratings can occur, and to prevent abuse by users or their friends, we need to introduce several mechanisms.

First, let’s discuss about the system components:

**CollaborationRegistry Contract:**

* This contract manages the registration and verification of collaborations between two parties. It stores details of each collaboration, including the public keys of both parties and a unique identifier for the collaboration.

**SBTRegistry Contract:**

* This contract handles the linkage and updating of SBT ratings. It verifies the validity of collaboration details through the CollaborationRegistry and ensures that only authorized parties can submit ratings using digital signatures.

### **Steps and Mechanisms**

#### **1. Collaboration Establishment**

* **Description:**
  + When a collaboration between party A and party B begins, it is registered in the CollaborationRegistry contract. This registration includes the public keys of both parties and a unique identifier for the collaboration.
* **Implementation:**
  + Party A and party B agree to collaborate and call a function on the CollaborationRegistry contract to register the collaboration. This function records both parties' public keys and generates a unique identifier for the collaboration.

#### **2. Verification Mechanism**

* **Description:**
  + To prevent unauthorized ratings, each rating submission must include a valid digital signature. The signature is linked to the other party's public key and the unique collaboration identifier, ensuring that the signature is specifically for that collaboration.
* **Implementation:**
  + When a rating is submitted, the SBTRegistry contract verifies the digital signature against the public key of the party being rated, the rating value, and the nonce. The signature ensures the authenticity of the rating request.

#### **3. Nonce for Replay Protection**

* **Description:**
  + Each rating submission includes a unique nonce to prevent replay attacks, ensuring that each rating can only be submitted once.
* **Implementation:**
  + The SBTRegistry contract maintains a mapping of used nonces and checks that the nonce has not been used before processing the rating submission.

#### **4. Ownership and Authorization**

* **Description:**
  + The contract is owned by a single address that has exclusive rights to call the updateRating function. This ensures that users cannot directly update their ratings.
  + The verifier is a designated address responsible for creating valid signatures.

4. Identity Verification with Zero Knowledge Proof:

In case of disputes, the prover needs to prove that they hold the private key and nonce such that the user address and public key are derived from them, without revealing the private key and nonce directly. We’ll use a Zero-Knowledge Proof (ZKP) to accomplish this. We can use a simplified form of ZKP for demonstration, such as a **commitment scheme combined with a non-interactive proof**.

#### **Steps for Construction & Verification of the Proof:**

1. Commitment Phase: The prover will create commitments to their private key and nonce using a commitment scheme.
2. Proof Generation: Compute the public key and user address using the committed private key and nonce.
3. Construct Proof of Knowledge: The prover needs to demonstrate that the computed public key and user address match the given ones, and that these values are derived from the committed private key and nonce.
4. Non-Interactive Proof: Use a Non-Interactive Zero-Knowledge Proof (NIZK) setup to create a proof that can be verified without interaction.
5. Verify the Proof: The verifier will verify the zero-knowledge proof that:

* The computed public key matches the provided public key.
* The computed user address matches the provided user address.

B. Investor Set:

In DeSC, it is possible for initiators to initiate a collaboration without having enough funds. Since an acceptor needs to see enough locked funds before she can accept an initiation, the initiator needs to find investment prior to initiation.

To facilitate collaborations where initiators lack sufficient funds, the system introduces a structured investment process. Initiators can secure investments through an Investor smart contract before proceeding with the collaboration initiation. This ensures that the acceptor sees enough locked funds before accepting the initiation. The process involves clear communication of project details, securing investments, and managing funds through multi-signature wallets to prevent fraud.

### **Process**

#### **1. Project Summary and Profile Creation**

* **Initiator Profile:**
  + The initiator summarizes their project on their profile. This profile is accessible to both potential investors and acceptors, providing essential details about the project.
  + The system includes a dashboard displaying profiles of all initiators, potential investors, and acceptors.

#### **2. Investment Acquisition**

* **Off-Chain Communication:**
  + Initiators reach out to investors through off-chain communication methods. Investors can visit initiators' profiles, check their SBT ratings, and review project descriptions.
  + The clarity and comprehensiveness of the project description are crucial to convincing potential investors.
* **Investment Deal:**
  + When an investor agrees to invest, they and the initiator formalize the investment deal via a function in the Investor smart contract.
  + The function includes parameters such as the amount of money being invested, the interest rate or other terms, and the payment timeline.

#### **3. Multi-Signature Wallet Creation**

* **Investor-Initiator Multisig Wallet:**
  + Upon accepting the deal, a multi-signature wallet (multisig wallet) is created involving the initiator and the investor(s).
  + This wallet requires multiple signatures for any transaction, ensuring that funds cannot be moved without mutual consent.

#### **4. Pooling Investments**

* **Multiple Investors:**
  + The initiator can make deals with multiple investors, creating a pool of investments. Each deal results in a new multisig wallet shared between the initiator and each investor.

#### **5. Collaboration Initiation**

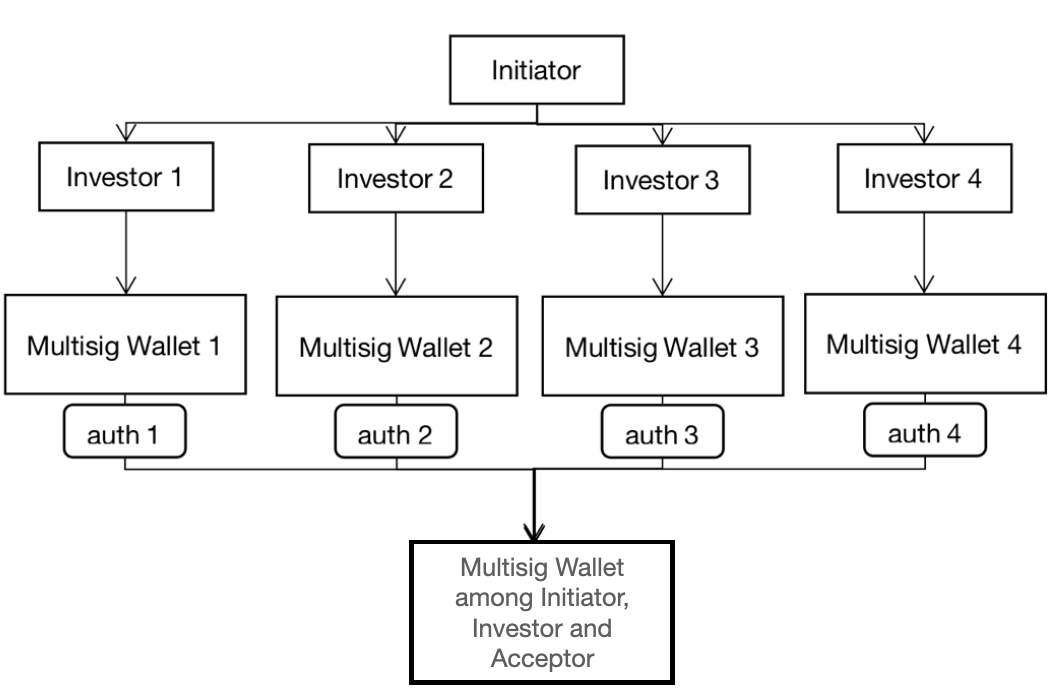
* **Reaching Out to Acceptors:**
  + After securing sufficient funds, the initiator contacts the acceptor via the system dashboard.
  + The initiator initiates the collaboration by calling the initiate contract. The acceptor can then accept the collaboration by calling the accept smart contract. This whole process is discussed further in the next section.
* **Initiator-Acceptor Multisig Wallet:**
  + Once the collaboration is accepted, another multisig wallet is created between the initiator and the acceptor.

#### **6. Fund Transfer and Authentication**

* **Controlled Fund Transfer:**
  + Funds are transferred from the investor-initiator multisig wallets to the initiator-acceptor multisig wallet.
  + Each transfer requires authentication from the relevant parties (e.g., both the initiator and the investor must authenticate the transfer).
* **Security Measures:**
  + Multi-signature wallets ensure that funds can only be moved with consent from all involved parties, preventing any single party from running away with the money.
  + This layered authentication eliminates the chance of fraudulent activities and ensures that funds are used as intended.

### **Security Measures**

* **Investor Protection:**
  + Multi-signature wallets ensure that funds are locked and can only be used with mutual consent.
  + Detailed records of all transactions are maintained on-chain, providing transparency and traceability.
* **Preventing Fraud:**
  + Authentication from all parties is required for any fund transfer, preventing unauthorized access.
  + Interest rates and terms are clearly defined in the smart contracts, and funds are disbursed according to these terms.
  + Initiator and acceptor individually or even collectively cannot withdraw funds from their own multi-sig wallet before the collaboration finishes. The withdrawal of funds will further need agreement from the moderators.
* **Dispute Resolution:**
  + In case of disputes, the system includes mechanisms for resolving conflicts and ensuring fair treatment of all parties.
  + Funds can be returned to investors if the collaboration does not proceed as planned, based on the terms defined in the smart contracts.



In the given figure the process of fund transferring is explained. auth1, auth2, etc. here means authentications. For example, auth1 means that Initiator and Investor1 need to authenticate so that the fund from multisig wallet1 reaches the multisig wallet between initiator and acceptor. Once the fund leaves the multisig wallet1, its corresponding interest rate starts.

C. Initiation and Acceptance of Collaboration:

In the proposed system, an **Initiator**—typically an individual or entity seeking collaboration—has the ability to view potential **Acceptors** through a system dashboard. This dashboard functions as a central hub that displays profiles of all users willing to engage in collaborations.

### **Initiator's View of Potential Acceptors**

1. **Dashboard Overview:**
   * The dashboard provides an organized list of potential Acceptors, each profile highlighting key information such as:
     + **Public Key**: Unique identifier for the acceptor within the blockchain system.
     + **SBT Rating**: A Soulbound Token rating reflecting the acceptor's trustworthiness and reputation.
     + **Skillset or Specialization**: Areas of expertise or services offered by the acceptor.
     + **Project History**: A brief overview of past collaborations or projects the acceptor has been involved in.
     + **Availability Status**: Indicating whether the acceptor is currently open to new collaborations.
2. **Project Compatibility:**
   * Initiators can filter and search for Acceptors based on specific criteria such as expertise, rating, and past project experience, ensuring a good match for their project's needs.
3. **Profile Exploration:**
   * Detailed profiles may also include additional data like:
     + **Portfolio**: Examples of previous work or projects.
     + **Testimonials**: Feedback from previous collaborators.
     + **Collaboration Preferences**: Types of projects the acceptor is interested in.
4. **Off-Chain Communication:**
   * After reviewing the profiles, the Initiator can initiate off-chain communication with potential Acceptors to discuss project details, expectations, and terms of collaboration before formally initiating the process on-chain.

This comprehensive view ensures that the Initiator can make informed decisions about which Acceptor to approach, fostering transparency and trust within the system.

**Off-Chain Communication with Chatbot:**

To facilitate efficient and secure off-chain communication between the Initiator and potential Acceptors, DeSC integrates a chatbot—a centralized, end-to-end texting application within the DeSC front-end, governed by back-end application programs.

1. Centralized Server and Security:
   * While the chatbot operates through a centralized server, ensuring swift and reliable messaging, the final consensus of the collaboration will be deployed on the blockchain. This ensures that the centralization of the chatbot does not compromise the decentralized nature of the overall system.
   * Accessibility to the chatbot will be highly restricted, maintaining security and privacy for the users.
2. Communication Flow:
   * DeSC caches all (initiator-acceptor-collaboration) pairs for specific initiators and acceptors on the server. This is achieved by storing signatures provided by both initiators and acceptors, along with the collaboration ID.
   * For any logged-in user, the chatbot will only display collaborations they are involved in, ensuring focused and relevant communication.
   * Communication between an initiator and an acceptor can only commence after the acceptor has formally accepted the collaboration. This prevents unsolicited messaging and ensures that all conversations are pertinent to active projects.
3. End-to-End Encryption:
   * To ensure the secrecy and privacy of user communication, the chatbot will employ an end-to-end encryption mechanism.
   * During the user setup phase, a public key-private key pair is generated. These asymmetric keys are used to create a symmetric key for faster communication between the initiator and acceptor.
   * This encryption strategy ensures that all messages are secure and can only be read by the intended recipient.

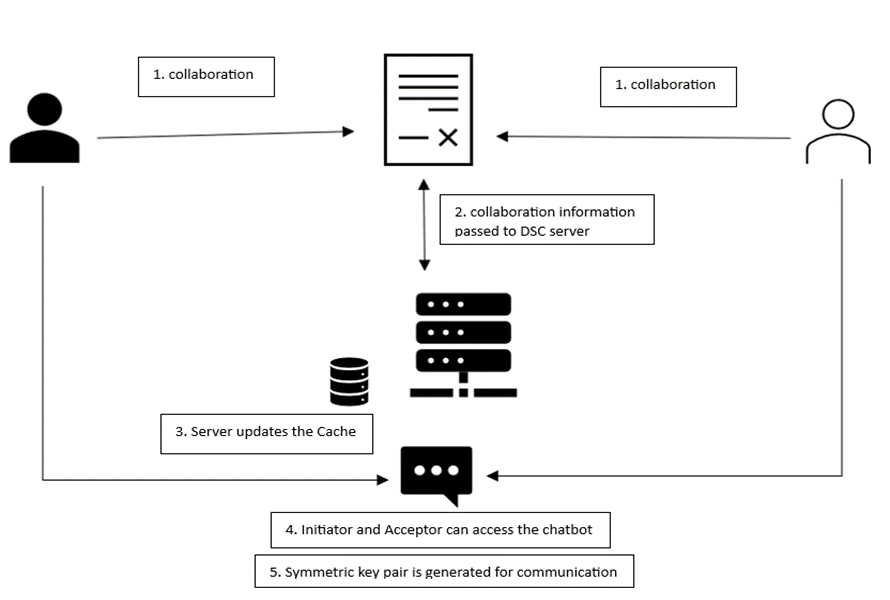


Fig: Chatbot explained

**Process Workflow:**

1. Project Summary and Dashboard Display:

* The Initiator summarizes their project in their profile. This profile is displayed on the system dashboard, visible to both acceptors and investors.

2. Viewing Potential Acceptors:

* Initiators can view potential acceptors on the dashboard. Profiles include:
  + Public Key (Unique Identifier)
  + SBT Rating (Reputation Score)
  + Skillset/Specialization
  + Project History
  + Availability Status

3. Initiating Contact:

* Off-chain communication is initiated through the chatbot after an acceptor shows interest by accepting the collaboration proposal.
* All interactions are logged and encrypted, ensuring privacy and security.

4. Centralized Chat Management:

* The centralized server manages chat sessions, ensuring that each user sees only the collaborations they are part of.
* Signatures and collaboration IDs are stored on the server to manage and verify active chats.

5. Symmetric Key Generation:

* Using the asymmetric key pair, a symmetric key is generated for each chat session to ensure fast and secure communication.

6. Blockchain Deployment:

* Once both parties reach a consensus through the chatbot, the agreement is signed by both the initiator and acceptor.
* The finalized collaboration agreement is deployed on the blockchain, ensuring an immutable and transparent record of the collaboration.

The Initiator and Acceptor smart contracts are crucial components of the DeSC system, facilitating the initiation and acceptance of collaborations. Below is an explanation of how these smart contracts will function, how they will be called, and how they will interact with each other.

#### **Initiator Smart Contract**

The Initiator smart contract is used by the initiator to propose a new collaboration and lock the necessary funds. This contract ensures that the collaboration details and the funds are securely managed.

**Key Functions:**

1. **Propose Collaboration:**
   * The initiator proposes a new collaboration by calling a function that includes collaboration details such as the project summary, the public key of the initiator, and the required funds.
2. **Lock Funds:**
   * The initiator locks the necessary funds in the smart contract to show commitment to the project. This ensures that the acceptor can trust the initiator's ability to pay for the collaboration.

#### **Acceptor Smart Contract**

The Acceptor smart contract is used by the acceptor to accept a proposed collaboration and initiate the collaboration process. This contract interacts with the Initiator contract to verify the details and ensure the funds are locked.

**Key Functions:**

1. **Accept Collaboration:**
   * The acceptor accepts the proposed collaboration by calling a function that verifies the collaboration details and ensures the required funds are locked.
2. **Initiate Collaboration:**
   * Once the collaboration is accepted, the acceptor can initiate the collaboration process, which involves unlocking the funds and beginning the collaboration.

### **Interaction Between Contracts**

1. **Proposing a Collaboration:**
   * The initiator calls proposeCollaboration on the Initiator contract to create a new collaboration proposal and lock the required funds.
2. **Accepting a Collaboration:**
   * The acceptor calls acceptCollaboration on the Acceptor contract, which interacts with the Initiator contract to verify the collaboration details and ensure the funds are locked.
3. **Initiating a Collaboration:**
   * After accepting the collaboration, the acceptor can call initiateCollaboration on the Acceptor contract to unlock the funds and start the collaboration process.

D. Moderator Set and Moderator Universe:

In this section we will talk about moderating a collaboration. We will assume that the initiator has already initiated the collaboration and the acceptor has accepted it. Since moderating itself is a complex algorithm, we cannot afford to make the process fully on-chain. Hence, as discussed earlier, we will allow all communication through the front-end based Chatbot and only the results/ conclusions to be stored in the blockchain. Since DeSC provides atomicity in collaboration, initiator and acceptor need to agree in a consensus. The consensus will finally be deployed in the blockchain for moderators to verify.

**Atomicity and Consensus:**

The primary objective for both the initiator and acceptor during a collaboration is to establish an atomicity consensus. This consensus represents a step-by-step division of the collaboration into smaller, verifiable units called "*atomic levels*." These levels ensure that each stage of the collaboration is assessed individually, rather than waiting until the entire project is completed. This approach provides greater oversight and ensures that both time and money are saved by catching potential issues early.

1. Establishing Atomicity Consensus:

The process begins off-chain, where the initiator and acceptor reach a mutual agreement regarding the atomic structure of their collaboration. Once the consensus is reached, the initiator submits the agreement on-chain by calling the consensus() function in the initiator smart contract. The initiator must provide both their digital signature and the atomicity consensus. The consensus is stored in a mapping variable, atom, which maps bytes32 values to bytes32. This format enforces a fixed-length restriction, requiring the consensus summary to be concise and to the point.

The reason for this limitation is twofold:

1. **Simplicity for Moderators**: Short summaries make it easier for moderators to quickly understand the key points of the collaboration and verify its progress.
2. **Efficiency**: The fixed length prevents excessively long entries, keeping the on-chain data streamlined.

Once the initiator submits the atomicity consensus, the acceptor must also provide their signature to confirm the agreement. Only after both parties sign off on the consensus is it granted access to the **Moderator Universe**, a system of moderators responsible for verifying each atomic level of the collaboration.

2. Atomic Levels and Moderator Verification:

Dividing the collaboration into atomic levels provides several advantages. Each level represents a discrete step in the project, allowing moderators to verify its accuracy and completion before moving to the next. This step-by-step verification ensures that potential issues are caught early, reducing the risk of wasting time or resources on a project that may have fundamental problems. Moderators are required to review the collaboration at each level, unlike traditional methods where verification happens only at the end.

This atomic approach provides a similar benefit to debugging multiple small functions in a program rather than troubleshooting a large block of code—each stage is easier to review and verify. Moreover, moderators are given clear, manageable tasks, reducing the chances of oversight or errors during review.

3. Flexibility in Moderator Selection:

One of the key advantages of this system is the flexibility it provides in moderator selection. The initiator and acceptor can choose a new set of moderators for each atomic level, or they can retain the same group throughout the project. This flexibility has two primary benefits:

1. **Security**: By selecting a different moderator set for each level, the initiator and acceptor can limit the knowledge any one set of moderators has about the full project. This ensures that no single group gains too much insight into the collaboration, thereby protecting sensitive details or intellectual property.
2. **Performance Evaluation**: The initiator and acceptor have the ability to evaluate the performance of moderators at each level. If they find a group to be underperforming or untrustworthy, they can choose a new set for the next atomic level, ensuring the collaboration is reviewed by the most capable and reliable moderators.

However, selecting a new moderator set for each level does come with drawbacks. First, it can be time-consuming to continuously vet and approve new moderators. Second, it incurs additional costs, as setting up a new moderator group requires updating the smart contract each time, which may lead to higher transaction fees.

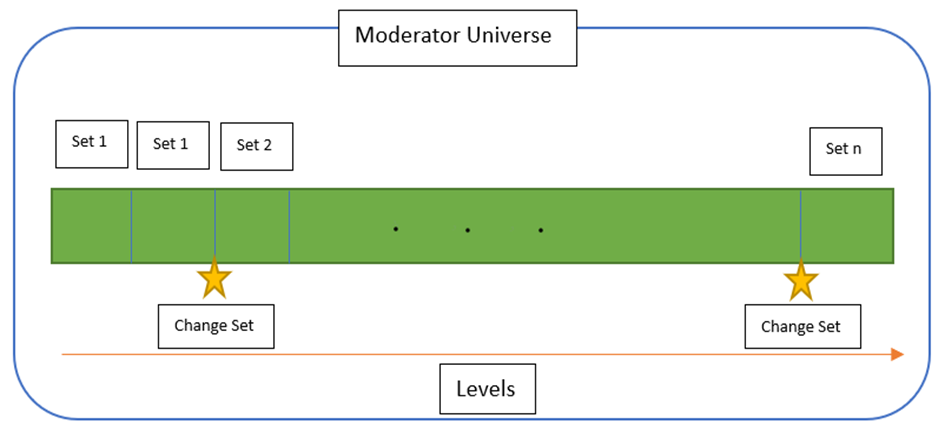


Fig: Moderator set per levels

**Moderator Universe and Moderator Set:**

The **Moderator Universe** is the complete pool of available moderators within the network. These moderators play a crucial role in validating different atomic levels of collaborations. From this universe, subsets are chosen to form a **Moderator Set** for each specific collaboration. A **Moderator Set** is a small group of randomly selected moderators responsible for validating a particular atomic level of a collaboration.

The size of the **Moderator Set** is determined by the initiator and acceptor. The process starts off-chain, where they agree upon this size via their communication in the chatbot. Once agreed, they call the initiate() function in the **Moderator Smart Contract**. This function requires the following parameters:

* **Size of the Moderator Set** (number of moderators).
* **Collaboration ID** (a unique identifier for the collaboration).

1. Trade-offs in Choosing the Moderator Set Size:

While maximizing decentralization would ideally involve having a large **Moderator Set** (even equal to the size of the **Moderator Universe** in the extreme case), there are practical trade-offs that come into play:

1. **Availability of Moderators**: Not all moderators in the universe may be available or willing to participate in a new collaboration at any given time. If an excessively large set is chosen, there might be delays in forming the set due to unavailability.
2. **Cost Considerations**: A larger moderator set incurs more costs, as each moderator’s participation needs to be compensated.
3. **Confidentiality and Privacy**: The larger the set, the more moderators are exposed to the details of a private or sensitive collaboration. In certain cases, keeping the moderator set smaller can help maintain privacy and reduce the risk of overexposure to the project.

2. Random Selection of Moderators:

To ensure fairness and decentralization, moderators in the set are selected randomly. This randomness serves multiple purposes:

* **Minimizing Bias**: Random selection helps prevent any one group of moderators from influencing each other or being selected based on known relationships.
* **Domain Diversity**: By selecting moderators randomly, it decreases the likelihood that moderators from similar domains or backgrounds end up reviewing the same collaboration level, which helps preserve objectivity.

Once the size of the moderator set is determined, the **Moderator Smart Contract** randomly selects moderators from the **Moderator Universe**. The selection process takes into account factors such as:

* **Current Availability**: Moderators who have indicated that they are active and willing to take on new tasks.
* **Previous Participation**: Avoiding the same moderators being repeatedly selected for multiple atomic levels of the same collaboration (unless the initiator and acceptor explicitly choose them).

3. Informing Moderators and Notifications:

After the random selection, the chosen moderators must be informed of their assignment. This can be achieved in two ways:

1. **Notification System**: The system can trigger an automated notification to selected moderators, either via email or through the platform's dashboard. Moderators will receive information about the collaboration ID (or a pseudonymous identifier to maintain privacy).
2. **Dashboard Updates**: Each moderator will have access to a personalized dashboard where they can view all the collaborations they have been selected to moderate. This dashboard will display key details, including the atomic level of the collaboration and any relevant deadlines.

Moderators can choose to accept or reject the invitation to moderate. If a moderator declines, the smart contract can automatically replace them with another available moderator from the universe.

4. Custom Invites to Specific Moderators:

While the default selection is random, there may be cases where the initiator and acceptor want to invite specific moderators due to their expertise. In such cases, the system allows them to send personal invitations to individual moderators. This invitation is done via a special, random link, which hides the actual **Collaboration ID** to maintain privacy. This approach ensures that moderators are only aware of the level they are moderating, not the full collaboration details.

The **Moderator Set** can be modified at each atomic level. The initiator and acceptor can retain the same moderators or request a new random selection for each subsequent level. This modular approach ensures both flexibility and security throughout the collaboration.

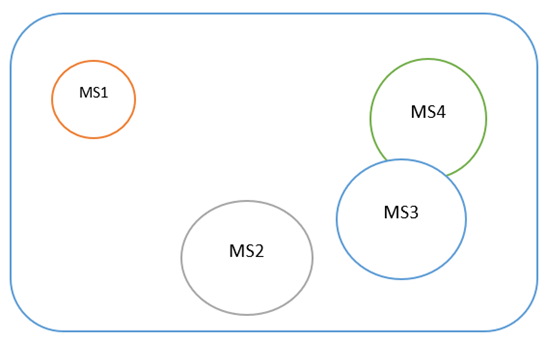


Fig: Moderator Universe and Moderator Set

**The Voting Phase:**  
The voting phase within the DeSC's Moderator Set works similarly to a Senate review. Moderators evaluate each level of the collaboration to ensure the consensus established between the **initiator** and the **acceptor** is valid. Their primary role is to identify any conflicts between the two parties, ensuring that both are adhering to the terms of the collaboration at the current atomic level. Here’s how the voting phase works, including the handling of conflicts and the **Motion for No Confidence (MNC)** process:

1. Positive Voting & Ignored Votes:

In the absence of conflicts, moderators can cast a **Positive Vote**, indicating that the atomic level of the collaboration is proceeding smoothly. However, moderators also have the option to **ignore voting**. Ignored votes are automatically counted as positive votes, which reduces the burden on moderators and encourages quick resolutions when things are progressing smoothly.

2. Conflict Identification & Initial Negative Vote:

When a conflict arises — for example, the **initiator** might be rejecting valid work from the **acceptor**, or the **acceptor** might not be fulfilling her obligations — a moderator has the option to cast an **Initial Negative Vote**. This signals that they believe there is a substantial issue with the collaboration.

3. Motion for No Confidence (MNC) Process:

If at least **1/5th of the moderators** vote negatively, a **Motion for No Confidence (MNC)** is triggered. This is a more serious escalation, and it represents a lack of faith in the collaboration proceeding successfully. The moderators who initiate the MNC process must stake their **SBT credit score** as part of the procedure, meaning that if their negative judgment turns out to be wrong, they risk losing credits. This discourages frivolous or biased MNC requests and ensures that moderators only raise an MNC if they are genuinely confident in their assessment.

* **MNC Success**: If the MNC is successful (i.e., the final vote affirms the initial negative votes), the moderators who initiated the MNC will receive a **boost in their SBT credit score**.
* **MNC Failure**: If the MNC fails (the majority votes against it), the initiating moderators will **lose SBT credits** as a penalty for their incorrect assessment.
* **Threshold Not Reached**: If the number of moderators supporting the MNC falls short of the 1/5th threshold, the MNC is **not called**, but no penalties are incurred by the moderators who initially voted negatively. This incentivizes moderators to vote negatively only when they are confident in the outcome, without the fear of losing points if others disagree.

4. Final Voting Phase:

When an MNC is triggered, all moderators are required to cast a **final vote**. This is done using the cast\_final\_vote() function in the **Moderator Smart Contract**. The result of this final vote is processed by the private count\_vote() function, which tallies all positive and negative votes to determine the outcome.

1. **If the MNC succeeds** (i.e., negative votes reach the majority), the moderators who supported the MNC are rewarded with a substantial **SBT credit boost**.
2. **If the MNC fails** (i.e., positive votes hold the majority), the MNC initiators lose credit, while other moderators who voted with the majority also receive a smaller **credit reward**. This mechanism rewards accuracy and penalizes votes that diverge from the consensus, ensuring that votes are cast carefully.
3. Moderators who voted **against the majority** (either way) will lose SBT credit points for being out of sync with the broader judgment, while those who align with the majority gain credit. The credit gained from aligning with the majority is smaller for non-MNC initiators, creating a **tiered reward system** that values both proactive conflict identification and accurate final voting.

5. Scenario Where MNC Is Not Called:

In the event where the **1/5th threshold** for an MNC is **not reached**, the MNC is not triggered. However, in this case:

* Moderators who cast **initial negative votes** are not penalized for their votes. This safeguard prevents moderators from fearing potential punishment for voting negatively, as long as their votes don’t trigger an MNC.
* The collaboration moves forward based on the majority of **positive votes**, and the process continues as normal. Since ignored votes count as positive votes, there is an implicit bias towards continuation, ensuring that collaborations are not halted unnecessarily due to a minority opinion.
* Moderators who **ignored the voting process** or voted positively will be seen as contributing to the successful completion of the atomic level and may receive **small credit rewards** for their participation, depending on how the final vote plays out.

6. Important Considerations for MNC:

**Impact on Final Voting**: Even though an MNC signals that at least 1/5th of the moderators have cast a negative vote, it doesn’t necessarily influence the outcome of the final vote. This is important because moderators are aware that the remaining 4/5th of the votes can still swing either way. This encourages **thoughtful and independent final votes**, preventing a bandwagon effect.

**Lazy Voting Risks**: Moderators who vote without properly scrutinizing the situation risk losing credit points if their votes end up on the wrong side of the majority. This is designed to prevent “lazy” voting where moderators just follow the initial MNC request without critically evaluating the situation.

### 7. Benefits of the Voting and MNC Process:

1. **Decentralized Decision-Making**: The system allows for fully decentralized conflict resolution through the moderator vote, ensuring that the collaboration process is transparent and community-driven.
2. **Checks and Balances**: By giving moderators the ability to raise an MNC, the system introduces an important check against potential abuse or error from either party in the collaboration.
3. **Incentives for Accuracy**: The use of SBT credit as both a reward and a penalty incentivizes moderators to vote carefully, promoting honest and diligent moderation.
4. **Safety in Numbers**: The threshold system ensures that MNCs are only called when there is genuine concern, avoiding disruption of the collaboration based on minority opinions.

This process ensures that collaborations are thoroughly moderated without excessive delays, and moderators are incentivized to participate in a fair and diligent manner.

**Section 5: Workflow**

In this section we will talk about the workflow of the collaboration.

Before going any further with the workflow, here are a few things that we should note:

1. We will assume that the initiator already has enough funds locked and thus do not need any funds from the investor set.
2. We will exclude the functionalities related to multisig wallets out of this workflow discussion.
3. We will exclude the functionalities related to gain and loss of SBT ratings and other profit (fiat or token) by moderators out of this workflow discussion.

### Step 1: Initiator and Acceptor Begin Discussions

The workflow begins with the Initiator proposing a project or collaboration to the Acceptor. This phase involves initial communication via a secure, encrypted chatbot. The chatbot provides a safe space for both parties to discuss and negotiate terms and requirements without interference. This conversation ensures both parties are aligned on the goals and expectations of the project before proceeding.

### Step 2: Defining Atomic Levels and Moderator Sets

Once the Initiator and Acceptor agree on the collaboration details, they proceed to create atomic levels of the collaboration. These atomic levels divide the project into smaller, verifiable milestones, providing incremental checkpoints to ensure the work progresses according to plan.

Moderators from the **Moderator Universe** are randomly selected, with the number of moderators adjusted based on project complexity and privacy needs. These moderators are responsible for overseeing the collaboration, verifying each atomic level, and participating in dispute resolution if conflicts arise.

Once everything is in place, the **Moderator Set** is **notified** of their role in the collaboration.

### Step 3.1: Collaboration Begins

The collaboration officially starts once the terms, requirements, atomic levels, and moderators are in place. The project moves forward through its various atomic levels, with each moderator set verifying and approving the completion of individual atomic level.

As the collaboration progresses, moderators evaluate the work, casting **votes** to confirm whether the milestone (or atomic level) was successfully completed.

### Step 3.2: First Round of Voting and Potential Conflicts

At each atomic level, moderators engage in a **first round of voting**, casting **positive or negative votes** based on their assessment of the progress.

* If **no significant issues** are detected, the project proceeds to the next atomic level, or the collaboration finishes, if there is no more atomic level left to validate.
* If **at least 1/5th of the moderators cast negative votes**, a **Motion of No Confidence (MNC)** is triggered, escalating the dispute resolution process. This is designed to address substantial concerns raised by moderators and ensure project integrity.

### Step 3.3: Motion of No Confidence (MNC) and Second Round of Voting

If an MNC is called, the collaboration enters a more serious phase of dispute resolution. In this phase:

* The system initiates a **second round of voting**.
* Moderators are asked to reconsider their votes and determine whether the project should proceed. This time, the vote has higher stakes:
  + If **51% or more of the moderators** cast negative votes, the MNC passes, and the atomic level in question is flagged for review by both the Initiator and Acceptor.

### Step 3.3.1: Reviewing the Questioned Atomic Level

Once an MNC passes:

* The Initiator and Acceptor must review the **specific atomic level** that triggered the negative votes (not the entire collaboration).
* They can choose to either:
  + **Adjust the terms and requirements** and continue the collaboration, incorporating moderator feedback, or
  + **End the process** if both parties agree that continuing is not feasible under the current conditions.

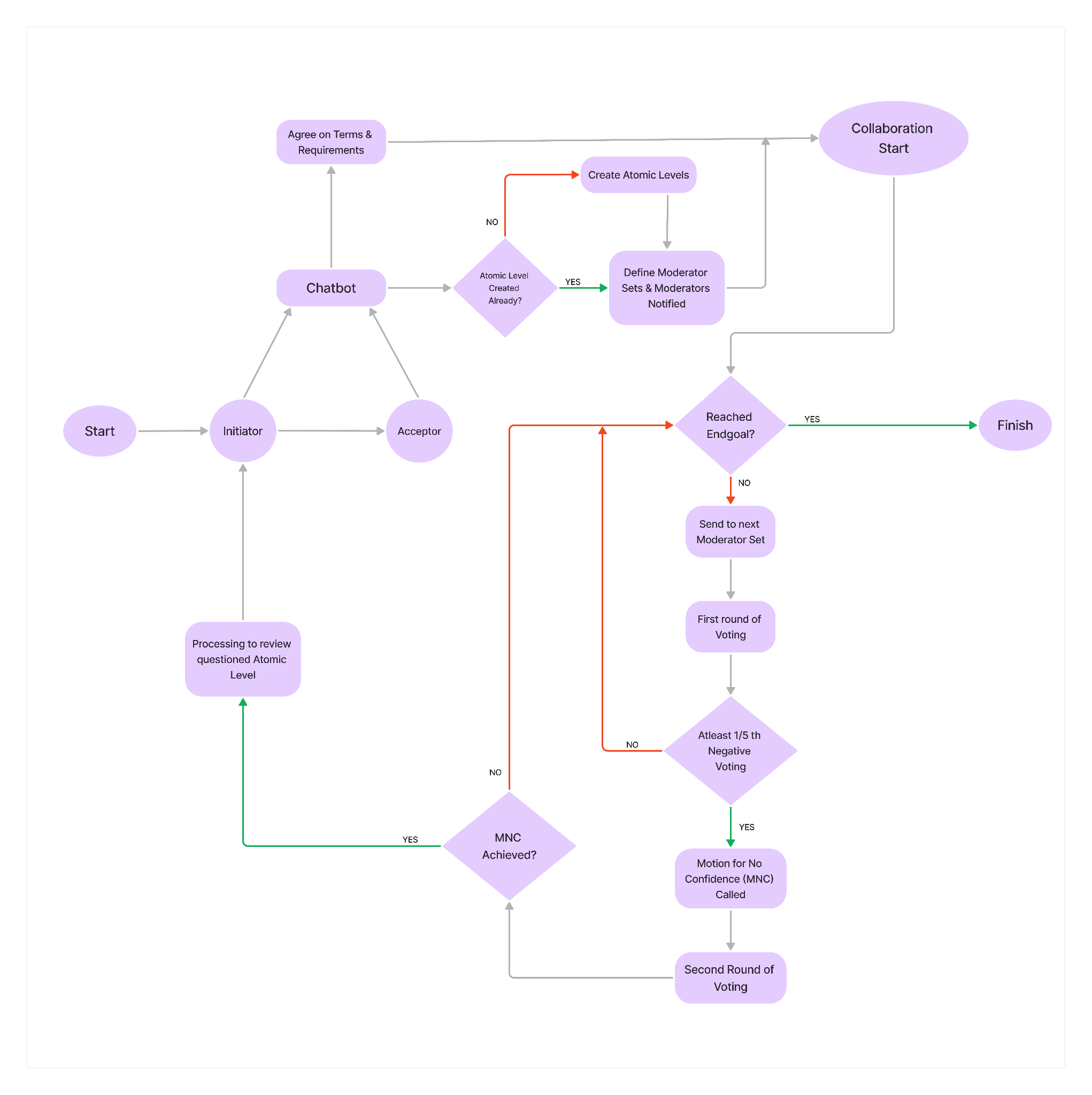
### Step 3.4: Failure to Achieve MNC

If the MNC fails (i.e., less than 51% negative votes), the project resumes its normal progression. The collaboration proceeds as though the MNC had never been called.

Step 4: Reaching the End Goal

When all atomic levels are complete, the system evaluates whether the project has reached its **end goal**. If the project successfully concludes:

* The collaboration process **finishes**, and the agreed-upon terms are fulfilled. This step includes fund distribution (managed via multi-signature wallets) and formal closure of the project.

****

**Fig: Workflow diagram**

**Section 6: Conclusion**

The Decentralized Secure Collaboration (DeSC) framework introduces a novel approach to managing collaborative projects by leveraging blockchain technology to ensure transparency, security, and accountability throughout the collaboration lifecycle. By integrating key concepts such as atomicity, multi-signature wallets, and decentralized moderation, DeSC addresses the critical challenges associated with trust and verification in collaborative environments.

DeSC establishes a comprehensive system for managing collaborations by utilizing smart contracts to handle various aspects of the collaboration process. Initiators and acceptors interact through a secure chat interface, negotiate terms, and formalize agreements on-chain. The framework supports a robust investment mechanism, where initiators can secure funding from investors before engaging with acceptors. The system’s modular design ensures that different components, including the Initiator Smart Contract, Acceptor Smart Contract, and Investor Smart Contract, function cohesively to facilitate seamless and secure collaborations.

Some specific characteristics:  
**Decentralized Management:** DeSC facilitates decentralized management of collaborations, ensuring that no single party has undue control over the process. The use of smart contracts and blockchain technology enables transparent and tamper-proof record-keeping.

**Multi-Signature Security:** The implementation of multi-signature wallets enhances the security of funds, requiring multiple parties to authenticate transactions. This mitigates the risk of fraud and ensures that funds are distributed only when all relevant parties agree.

**Dynamic Moderator Sets:** The framework supports dynamic selection of moderator sets for each atomic level of a collaboration. This flexibility allows initiators and acceptors to choose moderators based on their expertise and the sensitivity of the project.

**Advantages:**

1. **Enhanced Trust:** By utilizing blockchain’s immutable ledger, DeSC provides an unparalleled level of trust and transparency, reducing the likelihood of disputes and ensuring that all parties adhere to agreed terms.
2. **Security and Privacy:** The system employs end-to-end encryption for communication and multi-signature wallets for fund management, ensuring that sensitive information and funds are secure from unauthorized access.
3. **Efficient Dispute Resolution:** The atomicity concept, combined with the moderation process, allows for incremental verification of the collaboration’s progress. This approach simplifies dispute resolution and enhances oversight.
4. **Flexible Moderation:** The ability to select different moderator sets for each level provides flexibility and ensures that the most qualified individuals oversee critical stages of the collaboration.

**Future Goals:**

1. **Integration with Other Blockchain Networks:** Expanding DeSC’s compatibility with various blockchain networks could enhance its interoperability and broaden its adoption across different ecosystems.
2. **Advanced Moderation Algorithms:** Developing more sophisticated algorithms for moderator selection and voting could further optimize the moderation process and improve the accuracy of consensus.
3. **Scalability Enhancements:** Implementing scalability solutions to handle larger volumes of collaborations and transactions will ensure that DeSC remains efficient and responsive as adoption grows.
4. **User Experience Improvements:** Continuously refining the user interface and experience, particularly within the chatbot and dashboard components, will enhance usability and facilitate smoother interactions for all participants.

In conclusion, DeSC represents a significant advancement in the realm of decentralized collaboration. Its innovative use of blockchain technology and smart contracts paves the way for more secure, transparent, and efficient collaborative processes. As the system evolves, it promises to offer even greater value and functionality, contributing to the ongoing development of decentralized applications and ecosystems.

**Section 7: Appendix**

**Variables**:

mapping (address => mapping (string => (string, string))) public addressSBTLinker;

**Solidity Classes**:

**CollaborationRegistry Contract:**

contract CollaborationRegistry {

struct Collaboration {

bytes32 partyA;

bytes32 partyB;

bool isValid;

}

mapping(bytes32 => Collaboration) public collaborations;

event CollaborationRegistered(bytes32 indexed collaborationId, bytes32 partyA, bytes32 partyB);

// Function to register a new collaboration

function registerCollaboration(bytes32 collaborationId, bytes32 partyA, bytes32 partyB) external {

require(collaborations[collaborationId].isValid == false, "Collaboration already exists");

collaborations[collaborationId] = Collaboration({

partyA: partyA,

partyB: partyB,

isValid: true

});

emit CollaborationRegistered(collaborationId, partyA, partyB);

}

// Function to retrieve collaboration details

function getCollaboration(bytes32 collaborationId) external view returns (bytes32, bytes32) {

require(collaborations[collaborationId].isValid, "Invalid collaboration");

Collaboration storage collaboration = collaborations[collaborationId];

return (collaboration.partyA, collaboration.partyB);

}

}

**SBTRegistry Contract:**

import "./CollaborationRegistry.sol";

contract SBTRegistry {

CollaborationRegistry collaborationRegistry;

address private owner;

address private verifier;

mapping(bytes32 => uint256) private sbtRatings;

mapping(bytes32 => bool) private usedNonces;

event RatingUpdated(bytes32 indexed publicKey, uint256 rating);

modifier onlyOwner() {

require(msg.sender == owner, "Not authorized");

\_;

}

constructor(address \_collaborationRegistry, address \_verifier) {

owner = msg.sender;

collaborationRegistry = CollaborationRegistry(\_collaborationRegistry);

verifier = \_verifier;

}

function updateRating(bytes32 collaborationId, bytes32 publicKey, uint256 rating, bytes32 nonce, bytes memory signature) external onlyOwner {

require(!usedNonces[nonce], "Nonce already used");

(bytes32 partyA, bytes32 partyB) = collaborationRegistry.getCollaboration(collaborationId);

if (publicKey == partyA) {

require(verifySignature(partyB, rating, nonce, signature), "Invalid signature");

} else if (publicKey == partyB) {

require(verifySignature(partyA, rating, nonce, signature), "Invalid signature");

} else {

revert("Invalid public key for this collaboration");

}

sbtRatings[publicKey] = rating;

usedNonces[nonce] = true;

emit RatingUpdated(publicKey, rating);

}

function getRating(bytes32 publicKey) external view returns (uint256) {

return sbtRatings[publicKey];

}

function hasRating(bytes32 publicKey) external view returns (bool) {

return sbtRatings[publicKey] != 0;

}

function verifySignature(bytes32 publicKey, uint256 rating, bytes32 nonce, bytes memory signature) internal view returns (bool) {

bytes32 messageHash = keccak256(abi.encodePacked(publicKey, rating, nonce));

bytes32 ethSignedMessageHash = toEthSignedMessageHash(messageHash);

return recoverSigner(ethSignedMessageHash, signature) == verifier;

}

function toEthSignedMessageHash(bytes32 hash) internal pure returns (bytes32) {

return keccak256(abi.encodePacked("\x19Ethereum Signed Message:\n32", hash));

}

function recoverSigner(bytes32 ethSignedMessageHash, bytes memory signature) internal pure returns (address) {

(uint8 v, bytes32 r, bytes32 s) = splitSignature(signature);

return ecrecover(ethSignedMessageHash, v, r, s);

}

function splitSignature(bytes memory sig) internal pure returns (uint8, bytes32, bytes32) {

require(sig.length == 65, "Invalid signature length");

bytes32 r;

bytes32 s;

uint8 v;

assembly {

r := mload(add(sig, 32))

s := mload(add(sig, 64))

v := byte(0, mload(add(sig, 96)))

}

return (v, r, s);

}

}

**Investor contract:**

import "./CollaborationRegistry.sol";

contract Investor {

struct Investment {

bytes32 collaborationId;

address initiator;

address investor;

uint256 amount;

uint256 interestRate;

uint256 timeline;

bool active;

}

struct MultisigWallet {

address initiator;

address investor;

uint256 balance;

bool exists;

}

CollaborationRegistry public collaborationRegistry;

mapping(uint256 => Investment) public investments;

mapping(address => MultisigWallet) public multisigWallets;

uint256 public investmentCounter;

event InvestmentCreated(uint256 investmentId, bytes32 collaborationId, address initiator, address investor, uint256 amount, uint256 interestRate, uint256 timeline);

event FundDeposited(address multisigWallet, uint256 amount);

event FundWithdrawn(address multisigWallet, uint256 amount);

constructor(address \_collaborationRegistry) {

collaborationRegistry = CollaborationRegistry(\_collaborationRegistry);

}

function createInvestment(bytes32 \_collaborationId, address \_investor, uint256 \_amount, uint256 \_interestRate, uint256 \_timeline) public returns (uint256) {

(bytes32 partyA, bytes32 partyB) = collaborationRegistry.getCollaboration(\_collaborationId);

require(partyA == keccak256(abi.encodePacked(msg.sender)) || partyB == keccak256(abi.encodePacked(msg.sender)), "Sender not part of collaboration");

investmentCounter++;

investments[investmentCounter] = Investment({

collaborationId: \_collaborationId,

initiator: msg.sender,

investor: \_investor,

amount: \_amount,

interestRate: \_interestRate,

timeline: \_timeline,

active: true

});

emit InvestmentCreated(investmentCounter, \_collaborationId, msg.sender, \_investor, \_amount, \_interestRate, \_timeline);

return investmentCounter;

}

function createMultisigWallet(address \_initiator, address \_investor) public {

require(multisigWallets[msg.sender].exists == false, "Multisig wallet already exists");

multisigWallets[msg.sender] = MultisigWallet({

initiator: \_initiator,

investor: \_investor,

balance: 0,

exists: true

});

}

function depositFunds() public payable {

require(multisigWallets[msg.sender].exists == true, "Multisig wallet does not exist");

multisigWallets[msg.sender].balance += msg.value;

emit FundDeposited(msg.sender, msg.value);

}

function withdrawFunds(uint256 \_amount) public {

MultisigWallet storage wallet = multisigWallets[msg.sender];

require(wallet.exists == true, "Multisig wallet does not exist");

require(wallet.balance >= \_amount, "Insufficient balance");

if (msg.sender == wallet.initiator || msg.sender == wallet.investor) {

wallet.balance -= \_amount;

payable(msg.sender).transfer(\_amount);

emit FundWithdrawn(msg.sender, \_amount);

} else {

revert("Not authorized");

}

}

function verifyInvestment(uint256 \_investmentId) public view returns (bool) {

Investment memory investment = investments[\_investmentId];

return investment.active && investment.amount > 0;

}

}

**Initiator:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract Initiator {

struct CollaborationProposal {

bytes32 collaborationId;

address initiator;

uint256 requiredFunds;

bool isLocked;

}

mapping(bytes32 => CollaborationProposal) public proposals;

event CollaborationProposed(bytes32 indexed collaborationId, address indexed initiator, uint256 requiredFunds);

function proposeCollaboration(bytes32 \_collaborationId, uint256 \_requiredFunds) external payable {

require(msg.value == \_requiredFunds, "Incorrect funds provided");

proposals[\_collaborationId] = CollaborationProposal({

collaborationId: \_collaborationId,

initiator: msg.sender,

requiredFunds: \_requiredFunds,

isLocked: true

});

emit CollaborationProposed(\_collaborationId, msg.sender, \_requiredFunds);

}

}

**Acceptor:**// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

import "./Initiator.sol";

contract Acceptor {

Initiator public initiatorContract;

constructor(address \_initiatorContract) {

initiatorContract = Initiator(\_initiatorContract);

}

event CollaborationAccepted(bytes32 indexed collaborationId, address indexed acceptor);

function acceptCollaboration(bytes32 \_collaborationId) external {

(, address initiator, uint256 requiredFunds, bool isLocked) = initiatorContract.proposals(\_collaborationId);

require(isLocked, "Funds are not locked for this collaboration");

emit CollaborationAccepted(\_collaborationId, msg.sender);

}

function initiateCollaboration(bytes32 \_collaborationId) external {

(, address initiator, uint256 requiredFunds, bool isLocked) = initiatorContract.proposals(\_collaborationId);

require(isLocked, "Funds are not locked for this collaboration");

// Logic to initiate the collaboration

// Unlocking funds and starting collaboration process

}

}

**constructor Initiation (bool investor)**

{

if (investor == true)

{

call investor\_initiation(useraddress);

}

else

{

call acceptor\_initiation(useraddress);

}

}

**function investing(address self\_address, address initiator\_address)**

{

var total\_amount;

var rate;

var timeline;

//setters for the variables

//deploy investment amount to multisig wallet

}

contract Moderator {

mapping(bytes32 => bytes32) public atom;

function consensus(bytes32 collab) public {

atom [level\_info] = collab;

}

}

Moderator smart contract:

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract ModeratorSmartContract {

struct Moderator {

bool hasVoted;

bool vote; // true for positive, false for negative

}

struct Collaboration {

address initiator;

address acceptor;

uint256 moderatorSetSize;

mapping(address => bool) moderators;

mapping(address => Moderator) votes;

bytes32 atomicityConsensus;

bool consensusReached;

bool MNC; // Motion for No Confidence

uint256 positiveVotes;

uint256 negativeVotes;

}

mapping(bytes32 => Collaboration) public collaborations;

event ConsensusSubmitted(bytes32 indexed collaborationId, address initiator, address acceptor);

event ModeratorSetInitiated(bytes32 indexed collaborationId, uint256 size);

event VoteCast(bytes32 indexed collaborationId, address moderator, bool vote);

event MNCTriggered(bytes32 indexed collaborationId);

event FinalVoteTallied(bytes32 indexed collaborationId, uint256 positiveVotes, uint256 negativeVotes, bool MNCResult);

// Function to submit consensus by both initiator and acceptor

function consensus(bytes32 collaborationId, bytes32 atomicityConsensus, bytes memory initiatorSig, bytes memory acceptorSig) external {

Collaboration storage collaboration = collaborations[collaborationId];

require(msg.sender == collaboration.initiator || msg.sender == collaboration.acceptor, "Only initiator or acceptor can call this.");

require(collaboration.consensusReached == false, "Consensus already reached.");

// Both initiator and acceptor must submit their signatures

if (collaboration.atomicityConsensus == "") {

collaboration.atomicityConsensus = atomicityConsensus;

} else {

require(collaboration.atomicityConsensus == atomicityConsensus, "Consensus does not match.");

collaboration.consensusReached = true;

emit ConsensusSubmitted(collaborationId, collaboration.initiator, collaboration.acceptor);

}

}

// Function to initiate the moderator set, must be agreed by both initiator and acceptor

function initiate(bytes32 collaborationId, uint256 moderatorSetSize, bytes memory initiatorSig, bytes memory acceptorSig) external {

Collaboration storage collaboration = collaborations[collaborationId];

require(msg.sender == collaboration.initiator || msg.sender == collaboration.acceptor, "Only initiator or acceptor can call this.");

require(collaboration.moderatorSetSize == 0, "Moderator set already initiated.");

collaboration.moderatorSetSize = moderatorSetSize;

emit ModeratorSetInitiated(collaborationId, moderatorSetSize);

}

// Function for a moderator to cast an initial vote (positive or negative)

function cast\_initial\_vote(bytes32 collaborationId, bool vote) external {

Collaboration storage collaboration = collaborations[collaborationId];

require(collaboration.moderators[msg.sender], "Only assigned moderators can vote.");

require(!collaboration.votes[msg.sender].hasVoted, "Moderator already voted.");

collaboration.votes[msg.sender].hasVoted = true;

collaboration.votes[msg.sender].vote = vote;

// Count votes

if (vote) {

collaboration.positiveVotes += 1;

} else {

collaboration.negativeVotes += 1;

}

emit VoteCast(collaborationId, msg.sender, vote);

// Check if MNC should be called

if (collaboration.negativeVotes \* 5 >= collaboration.moderatorSetSize) {

collaboration.MNC = true;

emit MNCTriggered(collaborationId);

}

}

// Function to cast the final vote after MNC is triggered

function cast\_final\_vote(bytes32 collaborationId, bool vote) external {

Collaboration storage collaboration = collaborations[collaborationId];

require(collaboration.MNC, "MNC not triggered.");

require(collaboration.moderators[msg.sender], "Only assigned moderators can vote.");

require(!collaboration.votes[msg.sender].hasVoted, "Moderator already voted.");

collaboration.votes[msg.sender].hasVoted = true;

collaboration.votes[msg.sender].vote = vote;

if (vote) {

collaboration.positiveVotes += 1;

} else {

collaboration.negativeVotes += 1;

}

emit VoteCast(collaborationId, msg.sender, vote);

count\_vote(collaborationId);

}

// Private function to count votes and determine the result of MNC

function count\_vote(bytes32 collaborationId) private {

Collaboration storage collaboration = collaborations[collaborationId];

require(collaboration.MNC, "No MNC to resolve.");

if (collaboration.negativeVotes > collaboration.positiveVotes) {

// MNC Passed, collaboration failed

emit FinalVoteTallied(collaborationId, collaboration.positiveVotes, collaboration.negativeVotes, true);

} else {

// MNC Failed, collaboration continues

emit FinalVoteTallied(collaborationId, collaboration.positiveVotes, collaboration.negativeVotes, false);

}

}

// Utility function for adding a moderator to the collaboration

function addModerator(bytes32 collaborationId, address moderator) external {

Collaboration storage collaboration = collaborations[collaborationId];

require(msg.sender == collaboration.initiator || msg.sender == collaboration.acceptor, "Only initiator or acceptor can add moderators.");

require(!collaboration.moderators[moderator], "Moderator already added.");

collaboration.moderators[moderator] = true;

}

// Utility function to retrieve votes for a collaboration (public view)

function getVotes(bytes32 collaborationId) external view returns (uint256 positiveVotes, uint256 negativeVotes) {

Collaboration storage collaboration = collaborations[collaborationId];

return (collaboration.positiveVotes, collaboration.negativeVotes);

}

}

**REFERENCES:**

[1] <https://www.omnicoreagency.com/instagram-statistics/>

[2] <https://www.statista.com/statistics/1250681/share-of-instagram-influencers-involved-in-fraud-worldwide/>

[3]<https://transparency.fb.com/data/community-standards-enforcement/fake-accounts/facebook/>

[4] <https://about.linkedin.com/transparency/community-report>

[5] <https://www.techtarget.com/searchsecurity/feature/LinkedIn-scams-fake-Instagram-accounts-hit-businesses-execs>

[6] <https://www.phishlabs.com/blog/fraud-impersonation-fuel-q2-increase-in-social-media-attacks/>

[7] <https://www.ftc.gov/news-events/data-visualizations/data-spotlight/2022/01/social-media-gold-mine-scammers-2021>

[8] <https://www.cbsnews.com/news/influencer-marketing-fraud-costs-companies-1-3-billion/>

[9] <https://www.businesswire.com/news/home/20180515005775/en/Global-Brand-Counterfeiting-Report-2018-2020---ResearchAndMarkets.com>

[10] <https://www.statista.com/statistics/379046/worldwide-retail-e-commerce-sales/>

[11] <https://www.oberlo.com/statistics/how-many-people-shop-online>

[12] <https://www.paymentsdive.com/news/e-commerce-fraud-to-hit-20-billion-2021-an-18-jump-from-prior-year/599312/>

[13] <https://www.statista.com/topics/9240/e-commerce-fraud/#topicHeader__wrapper>

[14] <https://www.verifi.com/wp-content/uploads/2015/03/Verifi_wp_How-Much-Are-Chargebacks-Really-Costing-You-011515.pdf>

[15] <https://risk.lexisnexis.com/insights-resources/research/us-ca-true-cost-of-fraud-study>

[16] <https://www.cybersource.com/content/dam/documents/en/cybersource-ecommerce-fraud-explained-ebook-2020.pdf>

[17] <https://chargebacks911.com/chargeback-fraud/>

[18] <https://chargebacks911.com/chargeback-stats/>

[19] <https://www.forbes.com/sites/louiscolumbus/2020/05/18/how-e-commerces-explosive-growth-is-attracting-fraud/>

[20] <https://www.mastercardservices.com/en/recovery-insights/chronicles-new-normal-cybersecurity>

[21] <https://www.juniperresearch.com/press/ecommerce-losses-online-payment-fraud-48bn>

[22] <https://nilsonreport.com/upload/content_promo/NilsonReport_Issue1209.pdf>

[23] <https://www.crowe.com/global/news/fraud-costs-the-global-economy-over-us$5-trillion>