Ruby: a private data management layer for Web 3.0 in a multichain world

Ruby Labs
info@ruby.io

Version 1.0.1

Abstract

Ruby protocol is a private data management framework for Web 3.0. It proposes and implements a privacy middle-layer interacting with multi-chain. It is a fine-grained private data access-control gateway across different entities and organizations both in the decentralized and traditional financial world. Built on functional encryption, Ruby protocol will be the true embodiment of the decentralization spirit of the cryptocurrency movement. All the encrypted data will be stored in a decentralized cloud such as IPFS. Decentralized functional encryption will be adopted to satisfy the users' privacy needs. Moreover, our policy management layer will make sure the design of the underlying attribute and policy universe is well-tuned to the regulatory compliance requirement. Ruby will also design and implement a private payment scheme specifically tailored for the monetization of private data. Ruby protocol is not only the next-generation privacy protocol for Web 3.0 in a multi-chain world but is also ready to open a brand-new chapter for regulationcompliant decentralized financial service.

Contents

1 Introduction

1.1 What Problem Ruby Network Solves

Web 3.0, is an emerging new model for the internet economy that is calling for redistribution and democratization of value flowing on the internet. Web 3.0 revolution has brought us many interesting concepts and products. For instance, decentralized finance (DeFi) allows any user to have an access to sophisticated financial services without the need of opening a bank account. As the size of the cryptocurrency market grows, we expect many traditional institutions to join the DeFi market in the near future. A natural requirement of institutional money is regulatory compliance. Due to the sensitive nature of know-your-client (KYC) information, users, especially newcomers to DeFi are usually reluctant to give out their private information to the DeFi apps mostly due to the fear of losing control of their private data. On the other hand, the traditional financial (TradFi) system has gathered tremendous KYC information regarding its customers. How to leverage this private information while addressing the users' privacy concerns so that more TradFi users can join the sphere of DeFi has become a central challenge. As a matter of fact, many traditional institutions such as HSBC or Lloyds have already joined the open banking movement, in which consumers are able to grant their KYC data to third-party providers in a fine-grained manner. A private data sharing mechanism between TradFi institutions and DeFi apps that enables the users to have fine-grained access control on their private data will be in high demand as the regulatory compliance requirement for DeFi grows.

Another interesting concept that represents the spirit of Web 3.0, non-fungible token (NFT), first as a new way of monetizing digital artwork, has gradually been transformed into a new frontier of redefining decentralized identifiers (DID). NFT can play various roles in decentralized projects. For instance, numerous decentralized projects have issued NFTs to represent how involved a particular community member is with the project. An artist can issue an NFT as an exclusive membership token for his/her fans. In many cases, the ownership of an NFT also signifies one's identity or status. As a matter of fact, there is a growing trend using NFT as a premise for gated access to either digital or physical experience [nftGatedAccess]. For instance, imagine you wish to attend an NFT-gated online concert on a Metaverse platform. You could only attend the concert if you are either the owner of at least one Crypto Punk or at least two Bored Apes. Here the online concert could be replaced with an art exhibition in the Metaverse or a DAO zoom meeting, and it could also be a physical service such as renting a bike [pavemotors]. The access policy could be any logical formula that connects all kinds of NFT tokens.

Compared with traditional public-key encryption where the message receiver gets either nothing or all, functional encryption allows the message sender to precisely define what information on the encrypted message is allowed to be revealed to the receiver. In fact, functional encryption was originally proposed as a tool to enforce fine-grained access control over encrypted data [GPSW2006]. Therefore, it is a natural solution to all the aforementioned access control problems. Take the gated NFT case as an example, the access key to the online concert could be encrypted under the policy

"1 Crypto Punk & 2 Bored apes". Only those who have a secret key corresponding to at least two Bored apes or 1 Crypto Punk is able to decrypt the secret key entering the concert. Similarly, functional encryption can also serve as a handy tool for the private data-sharing platform between TradFi and DeFi.

In this context, Ruby will design and implement a fine-grained personal data management framework, which would serve as a second-layer/middleware protocol interacting with the multi-chain ecosystem. The framework will enable a data owner or an entity representing the owners to enforce a fine-grained access control policy over the encrypted private data using Functional Encryption (FE). The access control policy will be built into the smart contracts, and the relevant monetary transaction will also be executed via the smart contracts. This is why this solution is defined as a second-layer/middleware protocol.

The major technical contributions of this project will be:

- We build a private data management framework by leveraging the power of functional encryption and smart contracts. The proposed system will enable a fine-grained data management layer for the multi-chain world and Web 3.0, including a private data sharing platform between TradFi and DeFi, NFT-gated access, etc. A Fine-grained access control policy layer is proposed so that it can be built into the smart contracts to be deployed in multi-chain. The policy layer will be instrumental to satisfy the need of the regulatory compliance requirement and NFT-gated access control
- We devise an access control mechanism to ensure that legitimate buyers, on receiving the data owner's permission, can get access to the target data, be it the private KYC data in the case of private data sharing between TradFi and DeFi, or the access token of NFT-gated access system.
- We introduce a payment system, which will not only enforce payment privacy when needed but allow the accrue of value transferred among different parties in the system. We also introduce a review mechanism, with incentivization to buyers, to attract more sellers and buyers to our platform, and to motivate them to behave honestly.
- We employ a digital watermarking technique to guarantee the traceability of the encrypted data, which will add an additional guarantee on the authenticity of the users' private data.

1.2 Why Choose Functional Encryption

There are three relevant projects to Ruby: the first one is perhaps the Enigma project, a privacy protocol that enables the creation of decentralized applications (DApps) that guarantee privacy. The protocol Enigma is based on secure multi-party computation (MPC). The second one, Insights Network, is a data exchange protocol based on combining blockchain technology, Substrate module, and MPC. It is based on the EOS blockchain and a custom MPC system. The third one, NuCypher, is a cryptographic infrastructure for privacy-preserving applications. Its main technology is threshold proxy re-encryption and fully homomorphic encryption.

There are several different ways of implementing an MPC protocol: threshold homomorphic encryption, garbled circuit, and secret sharing. The general idea of MPC is to outsource private data (either in the form of secret shares or homomorphic encryption) to a few separate computing parties so that they can perform confidential computation over the encrypted data. Directly applying MPC to fine-grained data monetization is problematic in the sense that once the data is outsourced, the data owner does not retain any control over what type of computation can be performed by the computing party. In other words, individual privacy is now at the mercy of these computing parties, which is against the owner-centric ethos of fine-grained data management, where the access control policy should be defined by the data owner and enforced by the algorithm.

On the other hand, functional encryption was specifically proposed and tailored for enforcing fine-grained access control over encrypted data. Compared with traditional public-key encryption schemes that only allow the message receiver to either decrypt the whole data set or nothing, functional encryption allows the sender to determine exactly which part of the message can be decrypted by exactly which kinds of receivers. By allowing the data owner to define the access control policy, the owner has full control over what type of access the data purchaser can have over the encrypted data. The only decryption result the data querier will be able to retrieve is the predefined function evaluation.

1.3 Competitive Edge

Data privacy is always high on the agenda when it comes to the Crypto community. Since 2015, different projects have been working to improve that in three directions:

1. Anonymous currency, represented by Zcash, Monero, etc. They mainly meet the need of anonymous payment.
2. The privacy projects for crypto payment and transaction, represented by Tornado, zkSync, Starkware, Raze Network, etc. They mainly meet the privacy needs of smart contracts.
3. Data-based privacy computing projects, represented by Ruby Network, Oasis, and PlatON. They mainly meet the privacy computing needs of the data transaction market.

On the track of data privacy computing, Ruby's main competitors are Oasis, PlatON, Phala, Enigma, ARPA, etc. Ruby's main differentiated advantage lies in proposing a privacy computing solution based on user data attributes for Functional Encryption computing. Users can have fine-grained control of their data so that data buyers can obtain the computing results while the user's specific data content remains private.

	Encryption Tech	Value Position	Whether Fine-Grained	Built Data Marketplace	Owner Centric
Oasis	Trusted Execution Environment	Layer 1 public chain for encryption Apps	No	No	No

	Encryption Tech	Value Position	Whether Fine-Grained	Built Data Marketplace	Owner Centric
PlatON	The Mix of Multiparty Computation, Zero-Knowledge Proof, Verifiable Computing, Secret Sharing, Homomorphic Encryption	Layer 1 public chain for encryption Apps	No	No	No
Phala	Trusted Execution Environment	Layer 1 public chain for encryption Apps	No	No	No
Ruby	Functional Encryption	Layer 2 Application, focus on building data marketplace	Yes	Yes	Yes
Enigma	Secret Sharing and Multiparty Computation	Layer 1 public chain for encryption Apps	NO	NO	No
ARPA	Multiparty Computation	Layer 1 public chain for encryption Apps	No	No	No

The aforementioned projects are mostly based on cryptographic solutions. There also exist several projects, such as Mintlayer [Mintlayer] or lit protocol [litprotocol], that try to implement access control based on non-cryptographic solutions such as access control list (ACL). However, there are mainly two problems with ACL-based solutions:

- First, it does not provide confidentiality protection on the access control policy. On the other hand, functional encryption not only can enforce confidentiality requirements on the underlying message but also guarantee the secrecy of the access control policy.
- It is also hard to extend the access control policy to the real world beyond blockchain platforms given most of these ACLs can only be implemented as smart contracts that run on the smart contract platforms. In contrast, a functional-encryption-based solution can be adjusted to the access control policy that is defined on terms outside of the blockchain world. For instance, a user of TradFi can obtain secret keys corresponding to a certain KYC information from a TradFi institution to authenticate his/her identity to a DeFi app. In other words, functional encryption is a perfect tool to build a private data management middle-layer bridging the Web 2.0 world and Web 3.0.
- It is relatively hard to upgrade smart contracts corresponding to the ACL. A smart contract is a program executed in a highly decentralized environment. Any time a smart contract code is upgraded, it means all the participants in the system have to be aware of the upgrade and execute the necessary upgrade simultaneously, which could be a challenging task in a decentralized system. On the other hand, the revocation mechanism of attribute-based encryption or functional encryption can be achieved by adding an additional time dimension to the original attribute universe. All the keys will be naturally revoked once a specific time period passes.

1.4 Stakeholders

1) Validator

   In the Ruby protocol, Validators produce blocks and package transactions. They are responsible for running the nodes of the Ruby chain.

2) Nominator

   Nominators can delegate their RUBY tokens to Validators to gain staking rewards.

3) Data Owner

   Data Owner refers to users who provide data. They could be either the TradiFi users or institutions or the host of NFT gated events. The data owner gets benefits in different forms via fine-grained access control of data. For instance, the user might get discounted access to DeFi apps by using the private data-sharing program between TradFi and DeFi.

4) Data Querier

   Data Querier refers to users who wish to gain access to the data. It could be the TradiFi users' private KYC data or the access key to an NFT gated event. It publishes data calculation function f(x) out of its needs and purchases the data from Data Owner and Data Collector. The Buyer pays in accordance with the agreement to the Data Owner and Data Collector.

5) Data Collector

   Data Collector is responsible for the collection and processing of raw private data. This actor is optional, which is dependent on the concrete application scenario. For instance, the traditional financial institutions are the ones collecting private KYC information from the data owners, i.e., their clients.

6) RubyDAO Governor

   RubyDAO Governor is responsible for the governance of RubyDAO. It consists of the core developers of Ruby Protocol, popular Crypto KOLs, well-known big data experts and scholars, and privacy and regulatory-compliance legal experts. An important task of RubyDAO governor is to admit and register new key authorities.

2 Main layers

2.1 Policy management layer

The policy management layer defines the access control policy of the functional encryption scheme in the specific application scenarios. For instance, in the case of private data sharing between TradFi and DeFi, it will be the regulatory authorities and users who will be responsible for defining exactly what kind of private KYC information is allowed to leak to what kind of third-party DeFi apps. In the case of NFT-gated events, it will be the projects that issue NFTs that define the access control policy. The policy management layer will also specify exactly what type of policy information is privacy-sensitive and thus needs to be confidential.

2.2 Storage layer

The storage layer is responsible for storing the encrypted message under functional encryption scheme. We will adopt a chosen-ciphertext secure attribute-based encryption/functional encryption scheme to guarantee the integrity and authenticity of the underlying message. The digital watermarking technology will add an additional authenticity guarantee to the underlying message.

2.3 Credential issuance layer

The credential issuance layer is the anchor of the security of Ruby protocol. It will be responsible for setting up all the key authorities in the system. The key authorities will be responsible for verifying the user and distributing keys corresponding to attributes and policy depending on the types ABE/FE we adopt in the concrete application scenarios. The credential issuance layer will be run collectively by the RubyDAO governors and regulatory authorities.

2.4 Application layer

The application layer will interact with the blockchain apps, be it DeFi apps or NFT projects. It will be responsible for providing project requirements of blockchain apps and interacting with the entities responsible for the policy management to specify the privacy requirement of the blockchain apps. It will provide suitable APIs for these projects so that Ruby's privacy protection feature can be used in a plug-in-and-play manner.

3 Key features

The main features the Ruby system will guarantee are as follows:

- Fine-grained Data privacy: Our access control policy will be well-defined so that it is attuned to the specific application scenarios. A too general access control policy might render it unusable in reality given it might leak too little information on the users. On the other hand, a too specific access control policy will leak too much private information of the users. Therefore, the stakeholders of Ruby protocol will periodically update the policy management standard so that it can concord with the demand of real-world use cases and the latest regulation.
- Data quality and source assurance: Ruby guarantees the data coming from the right source and has the highest quality in the sense that it is not guaranteed to be authentic, but also up-to-date.
- Policy updateability: Since regulation and access control policy always change, our system is malleable and updateable so that all apps dependent on the policy can change swiftly once the policy changes.
- Attribute and policy revocability: Since the attribute value or attribute policy for a secret key might change with time, we should provide a mechanism to revoke the respective secret

key. For instance, consider the attribute “age”, a user’s age changes constantly as time goes by.

- ZKP and micropayment: Ruby employs the zkp and micropayment schemes in our system architecture to build a data monetization framework that can accrue the value transferred across different actors to the Ruby token.
- Multiple use cases: Ruby can serve as a fine-grained access control gateway for data flow in different application scenarios, including private data sharing platform between TradFi and DeFi, NFT-gated or DID-gated access control, and regulatory-compliant tokenized security issuance, etc.

4 Functional Encryption

4.1 Introduction

Function encryption was first proposed by Boneh, Sahai and Waters in 2011. This encryption algorithm broke the original ”All-or-Nothing” data query mode. The computing results of functional encrypted data can be obtained without revealing the plaintext content.

A classic Functional Encryption Scheme contains four algorithms, namely:

- FE.Setup: $(\mathrm{PK}, \mathrm{MSK}) \leftarrow \mathrm{Setup}(1^\lambda)$. The setting algorithm is used to generate the master key (MSK) and public key (PK)
- FE.KeyGen: $\mathrm{SK} \leftarrow \mathrm{KeyGen}(\mathrm{MSK}, k)$. The key generation algorithm is used to generate the key for the function f_k.
- FE.Enc: $C \leftarrow \mathrm{Encrypt}(\mathrm{PK}, x)$. Encryption algorithm is used to encrypt data.
- FE.Dec: $y \leftarrow \mathrm{Decrypt}(\mathrm{SK}, C)$. The decryption algorithm is used to safely calculate the function value $y = f_k(x)$.

The above is a general form of function encryption. In order to realize that users can achieve fine-grained control of their data, that is to say, the data owner can specify who can access the encrypted data and has complete control over the data, Ruby will use the Ciphertext-Policy ABE (CP-ABE) or Key-Policy ABE (KP-ABE) algorithm in the Functional Encryption algorithm to encrypt their personal data. The algorithms have to be CCA secure to guarantee the authenticity of the underlying message. The encryption algorithms will also have a revocation mechanism to ensure the access policy can be updated periodically. For more details on the encryption scheme, the interested readers are referred to our yellowpaper.

4.2 ABE Scheme

Based on the ABE scheme, Ruby will support the revocation and restoration of user attributes. The execution includes 7 steps:

1. **FE.Setup**: The initialization algorithm is a randomization one, and the initialization only generates the system public key PK and the system master secret key MSK.

2. **FE.Setup.Cancel**: The input parameter is *PK*, a prime number field P is generated, and *list=1* is calculated for each attribute att. The algorithm outputs the initialized P, *map<userGID,prime>* and *map<att,list>*. The list is not a common one but a large, prime number, which is used to record whether it has been revoked.
3. **FE.KeyGen**: By the set S provided by *PK*, *MK* and data requesters, the trusted authorization center generates the user secret key *UK* associated with the attribute set for the data requesters. Apply a prime number for the user in the prime number field, delete this prime from P to ensure that the primes obtained by different users are varied, and then store the prime number in the mapping table *map<userGID,prime>*.
4. **FE.Enc**: The input parameters of the encryption algorithm (randomization algorithm) are *PK*, the message to be encrypted *M*, the access control structure associated with the access policy, and the ciphertext based on attribute encryption is output.
5. **FE.Dec**: Decryption is a deterministic algorithm, executed by the data requester. Decryption is divided into two steps. The first step is to access the leaf node of the policy tree, let $i = att(x)$, where x represents the leaf node of the ciphertext policy access tree. If $i \in S$, the algorithm obtains the revocation list corresponding to this attribute list and modulo *list% prime*. If the value is not 0, it means that the user has not been revoked. If it is 0, it means that it has been revoked and the decryption is over. Step 2: After the verification of Step 1 is passed, the algorithm enters *UK* and ciphertext *M*. If the attribute set satisfies the access Strategy, the algorithm can successfully decrypt the ciphertext *M*.
6. **User.Attribute.Cancel**: *att* is the algorithm input parameter; *list* is the cancellation list corresponding to the attribute; and prime is the prime number prime corresponding to the user. When att of a user is cancelled, *DO* takes out the prime corresponding to the user and the list corresponding to the attribute from the mapping tables *map<user,prime>* and *map<att,list>* respectively, and calculates *list’=listprime*. The revocation list corresponding to the attribute *att* is updated to *list’*.
7. **User.Attribute.Recovery**: *att* is the algorithm input parameter; *list* is the cancellation list corresponding to the attribute; and *prime* is the prime number corresponding to the user. The cancellation of the user may be temporary. When the user re-owns the attribute *att*, *DO* takes out the *prime* number corresponding to the user and the *list* corresponding to the attribute from the mapping tables *map<user,prime>* and *map<att,list>* respectively, and calculate *list’=listprime*. The revocation list corresponding to the attribute *att* is updated to *list’*.

4.3 Attribute and Function Management

1) User management

Basic information owned by the user (minimum data set): GID (provided by CA or business side, but overall uniqueness must be guaranteed), CP-ABE user attributes (provided by the business side), unique prime number (provided by the CA user registration unit), private key (generated by

CA user private key generation unit).

The user submits information (GID, CP-ABE user attribute, unique prime number) to the CA, applies for registration and generates a user private key.

2) Modification of access control policy

In practical engineering applications, better choose the cancellation scheme with low complexity, low computing cost, and simple implementation. Therefore, the policy cancellation scheme adopted in this design is as follows: Revoke users through a fixed-length cancellation list record, so that the cancellation process does not need to update the secret keys of the system and related users, which can greatly reduce the computational burden caused by the cancellation.

3) Cancellation

User attribute cancellation: *att* is the algorithm input parameter; *list* is the cancellation list; *prime* is the prime number corresponding to the user prime. When the *att* of a user is cancelled, *DO* takes out the *prime* corresponding to the user and *list* corresponding to the attribute from the mapping tables *map<user, prime>* and *map<att, list>* respectively, and calculates *list'=listprime*. The cancellation list corresponding to the attribute *att* is updated to *list'*.

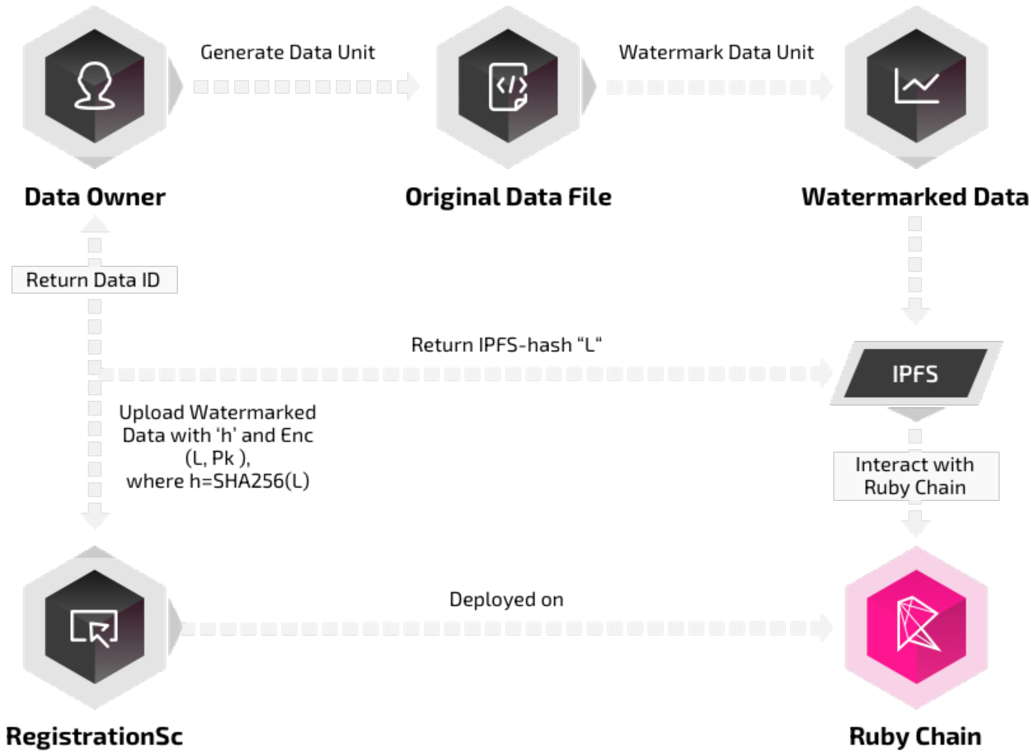
4) Recovery

User attribute restoration: *att* is the algorithm input parameter; *list* is the cancellation list corresponding to the attribute; *prime* is the prime number corresponding to the user. The cancellation of the user may be temporary. When the user re-owns the *att*, *DO* takes out the *prime* number and the cancellation *list* from the mapping tables *map<user,prime>* and *map<att,list>* respectively, and calculate *list'=listprime*. The cancellation list corresponding to the attribute *att* is updated to *list'*.

5 Data Copyright Protection

5.1 Structure Overview

In Ruby, only when the uniqueness and accuracy of the data source are fully guaranteed can the data copyright of the original owner of data be secured during the circulation of data. Therefore, Ruby has a Data Copyright Protection mechanism in place, including four functional modules: Registration, Watermarking, Storage, Registering Watermarked-data. The structure is shown in the figure below:

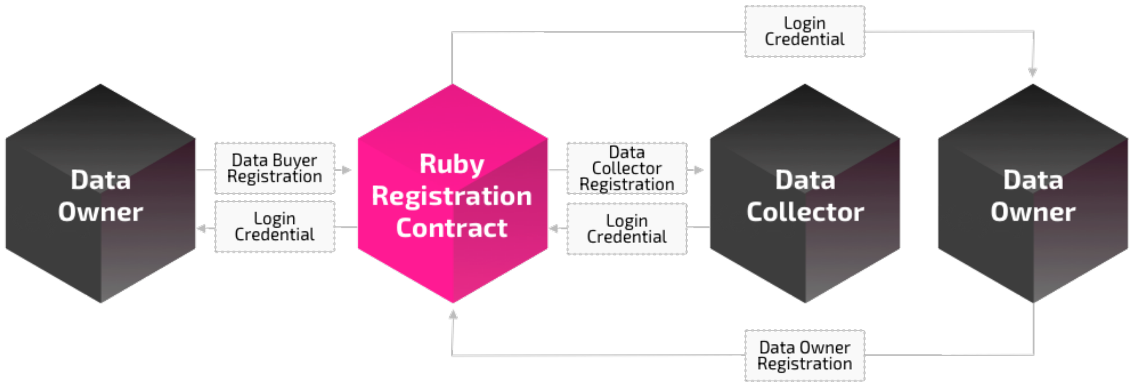


5.2 Registration

All participants of the Ruby protocol, including the Data Owner, the Data Collector, and the Data Buyer, must register first to join the market. In the registration, Ruby does not require users to fill in Personal Identity information in order to ensure user anonymity, but it needs to set basic ID information, such as user name, email address, information introduction, data introduction, the hash value of the data after running the data encryption algorithm, etc.

After the user fills in the information, they need to sign through the Ruby wallet and obtain the login credentials, which will be used for authentication later. The smart contract RegistrationSc stores participant details and generates login credentials. The login credentials include a unique identifier and password, which are stored on the Ruby blockchain.

The interaction logic between the Data Owner, the Data Collector, the Data Buyer and the RegistrationSc is shown in the figure below:



5.3 Watermarking

When a user completes registration, he needs to watermark the original data he owns using Ruby's watermarking technology tools, and embed the signature of his wallet address into the watermark information. This ensures the binding between the data and the user information.

Ruby will use the AHK algorithm (Agrawal et al., 2003; Agrawal & Kiernan, 2002) to add a watermark to the data. The watermark does not affect the value and use of the original content, and cannot be noticed or detected by the human perception system. It can only be extracted through a dedicated detector or reader. The watermark information includes the user ID registered by the Data Owner in the Ruby Data Marketplace, wallet address signature information, text with special meaning, etc. The information is used to identify the source, version, original author, owner, and issuer of data.

Given a database relationship $R\ (P,\ A_0,\ A_1\ ,\ A_2\ ....\ A_{v\text{-}1})$, where P is the main attribute, A_0 A_1 A_2 .... $A_{v\text{-}1}$ is the candidate attribute for marking.

The watermark embedding algorithm is shown in Algorithm 1 figure below:

The watermark detection algorithm is shown in the below Algorithm 2 figure:

Given a database relation$R\ (P,\ A_0,\ A_1\ ,\ A_2\ ....\ A_{v\text{-}1})$ where P is the primary key attribute and A_0 A_1 A_2 .... $A_{v\text{-}1}$ are candidate attributes (numeric) used for marking. The embedding and detection of watermarks are depicted in *Algorithms 1* and *2* respectively. In *Algorithm 1*, once a tuple r is found eligible for marking at step 2, the index i of the candidate attribute and its corresponding LSB position j are computed at steps 3 and 4, respectively. Step 5 invokes Mark function which flips the bit at jth LSB position in ith attribute depending on the hash of the private key concatenated with the primary key of the corresponding tuple. Observe that H and F are one-way hash and MAC functions, respectively.

The detection algorithm (*Algorithm 2*) considers a suspicious relation S and computes at step 6 the total number of tuples (denoted by *total-count*) already marked in the insertion algorithm. The '*Match*' function at step 7 checks whether the marked bits at LSB position are present and accordingly computes the total number of successful detection (denoted by *match-count*). If *match-count* is more than the threshold τ (computed based on α and *total-count*), then the watermark detection is considered as successful (shown in steps 10–13).

Algorithm 1: Watermark Insertion Algorithm

// The private key K and the parameter γ, ν and ξ are known only to the owner of the database.
// $1/\gamma$: Fracton of tuples marked.
// ν: Number of attributes available for marking.
// ξ: Number of least significant bits available for marking in an attribute.

for *tuple* $r \in R$ **do**
  if $F(r \cdot P)$ *mod* γ *equals 0* **then**
    attribute_index i = $F(r \cdot P)$ mod ν;
    bit_index j = $F(r, P)$ mod ξ;
    $r \cdot A_i = Mark(r \cdot P, r \cdot A_i, j)$;
  end
end
Mark(*primary_key pk, number v, bit_index j* **) return number**
  first_hash = $H(K \circ pk)$;
  if *first_hash is even* **then**
    set the j^{th} least significant bit of v to 0;
  else
    set the j^{th} least significant bit of v to 1;
  end
  return *v;*
return

Algorithm 2: Watermark Detection Algorithm

// α (where $0 < \alpha < 1$): Test significant level.
// τ: Minimum number of correctly marked tuples needed for detection.

total-count = match-count = 0;
for *tuple* $s \in S$ **do**
  if $F(s \cdot P)$ *mod* γ *equals 0* **then**
    attribute_index i = $F(s \cdot P)$ mod ν;
    bit_index j = $F(s, P)$ mod ν;
    total-count = total-count + 1;
    match-count = match-count + $Match(s \cdot P, s \cdot A_i, j)$;
  end
end
τ = Threhold(*total-count*, α);
if *matchcount* $\geq \tau$ **then**
  suspect piracy;
end
Match(*primary_key pk, number v, bit_index j* **) return int**
  first_hash = $H(K \circ pk)$;
  if *first_hash is even* **then**
    set the j^{th} least significant bit of v to 0;
  else
    set the j^{th} least significant bit of v to 1;
  end
  return *v;*
return

5.4 Storage

To make sure that the data stored by the user is secure, Ruby will upload and store user data processed by watermarking technology in the distributed storage network IPFS instead of a centralized cloud service. During the user upload process, Ruby Network will automatically tag and create attributes based on the content uploaded by the user to facilitate the operation of the Functional Encryption algorithm. When the upload is completed, IPFS will return to the user a hash value L of the uploaded data.

5.5 Register Watermarked-data

When the user obtains the hash value L, he/she needs to call RegistrationSc on the registration interface of the Ruby Data Marketplace to upload that. When the user completes the upload, he will get the unique data ID corresponding to the hash value L. At the same time, Ruby will run the SHA256 algorithm to encrypt L and store it on the Ruby block.

6 Use cases

6.1 Private KYC

It focuses on performing KYC such as biometric authentication without leaking private identity information. The functional encryption will be employed to guarantee that only controlled KYC and DID information is revealed to the entity users are trying to authenticate to. It will ensure user data and identity confidentiality while performing the KYC step.

6.2 Regulatory-compliant private data sharing between TradFi and DeFi

A user will be able to set up a Ruby account and connects with his bank account or centralized exchange accounts such as Binance. The relevant KYC information will be categorized and encrypted with the respective policy, such as “Vip-1 user” AND “Exchange: Binance” AND “DeFi app type: tier-1 DEX”, and uploaded to the storage layer. Only when the DeFi apps present a secret key corresponding to all these attributes will be able to decrypt more detailed user information. The Ruby API will establish an alliance between TradFi institutions and DeFi apps, and serve as an access control gateway to guarantee all the private data flow between both sides concords with the necessary regulation, etc.

Ruby Protocol’s privacy solution embodies the built-in attributes of data programmability, traceability, and verifiability, all of which will lead to greater adoption of distributed applications in auditing automation and compliance monitoring, and it will guarantee high assurance. Ruby Protocol’s solution can serve as a fundamental tool for TradFi to transit to DeFi.

6.3 NFT-gated or DID-gated event

People are turning everything into NFTs, be it videos, audios, JPGs, etc. It could represent your experience, your thoughts, your status, and sometimes NFT itself can be a secret key. NFT is also redefining decentralized identifiers (DID). In many cases, NFT is your identity. Therefore, access control based on NFT is a natural consequence. There are many traditional Internet companies such as Meta and Alibaba joining the race of NFT, which means the NFT-gated event could go well beyond the cryptocurrency world. There are numerous projects starting to use NFT to control the access to both online and offline events, be it online Metaverse events, hotels, bike renting, etc. There are almost infinite possibilities to combine NFT-gated access control with functional encryption. Even in the simplest form of whitelisting access control based on NFT, we could pick a power tool such as identity-based broadcast encryption (which is a special form of functional encryption) and provide an immediate solution. This is a really exciting space to watch.

6.4 Regulatory-compliant asset ownership

There is a growing trend to issue stocks or securities on the permissionless blockchain. For instance, CM-Equity AG issued Coinbase's stock (COIN) on the BNB smart chain recently, allowing the users to purchase tokenized versions of the stock. To ensure the regulatory compliance of security issuance, one could encrypt the access key to the said security under the predefined policy. For instance, the issued security might only be accessible to citizens of a center country. In this case, the access key should be encrypted under the allowed country so that any citizen with a key corresponding to the wrong country won't be able to decrypt the said security. Obviously, real-world regulations will be much more complex than this, but they could all be enforced in a similar manner using functional encryption.

Ruby's FE smart contracts are privacy-centric and underlie the relationship between asset owners and custodians. The bug-free and algorithmically correct code can ensure the ownership of any digital asset and its access control mechanisms can define who can own and transfer a specific asset in a regulatory compliant manner.

7 Consensus Mechanism

7.1 NPoS Consensus

The Ruby Network protocol is developed based on the Substrate blockchain development tool and belongs to the Polkadot ecology. So, Ruby will adopt the same NPoS (Nominated Proof of Stake) consensus mechanism as Polkadot.

7.2 Inflation Rate

In the Ruby network, the inflation rate (I) can be calculated by the below formula:

$$I = I_{NPoS} + I_{treasury} - I_{slahing} - I_{\textit{tx-fees}} - I_{burn}$$

I_{NPoS} refers to the inflation rewards distributed to validators and nominators. $I_{treasury}$ inflation refers to the rewards distributed to the Treasury. $I_{slahing}$ is the Slash penalty due to the malicious behaviour of the Validator, and $I_{tx\text{-}fees}$ refers to the destruction of transaction fees in the network. I_{burn} represents the repurchase and burning of Ruby Protocol revenues.

In the above formula, I_{NPoS} is a core variable, which is directly related to the Staking rate of the network. The sum of I_{NPoS} and $I_{treasury}$ is a fixed parameter of 10%. That means that the highest possible inflation rate in the Ruby Protocol network is 10%, but this does not mean that the Ruby network will always be in inflation.

In order to reduce the overall inflation rate of the Ruby network, Ruby will also regularly use part of the network's revenue for secondary market repurchase and destruction, which is represented in the formula as I_{burn}. This is accompanied by the default destruction mechanisms $I_{slahing}$ and $I_{tx\text{-}fees}$ in the NPoS consensus mechanism. When the ecosystem in Ruby is prosperous enough, transaction and repurchase & burning plans may offset the inflation rewards in the network, making the system deflationary.

7.3 NPoS Inflation

Under the NPoS consensus mechanism, the rewards for validators and nominators mainly come from the further issuance of tokens, which is also the main cause of network inflation. The inflation rate of Ruby Network is not adjusted by anyone manually but through an algorithm. The formula of the NPoS inflation model is shown below:

$$I_{NPoS}(x) = \begin{cases} I_0 + x\left(i_{ideal} - \frac{I_0}{\chi_{ideal}}\right) & \text{for } 0 < x \leq \chi_{ideal} \\ I_0 + (i_{ideal} \cdot \chi_{ideal} - I_0) \cdot 2^{(\chi_{ideal} - x)/d} & \text{for } \chi_{ideal} < x \leq 1 \end{cases}, \text{ and}$$

$$i(x) = I_{NPoS}(x)/x.$$

χ_{ideal} is the expected Staking rate, which represents the ideal Staking rate in the Ruby network; I_{ideal} is the expected annualized rate of return. R_0 represents the inflation rate when the Staking rate is 0 and that's the default inflation rate; d represents the decay rate of the inflation rate.

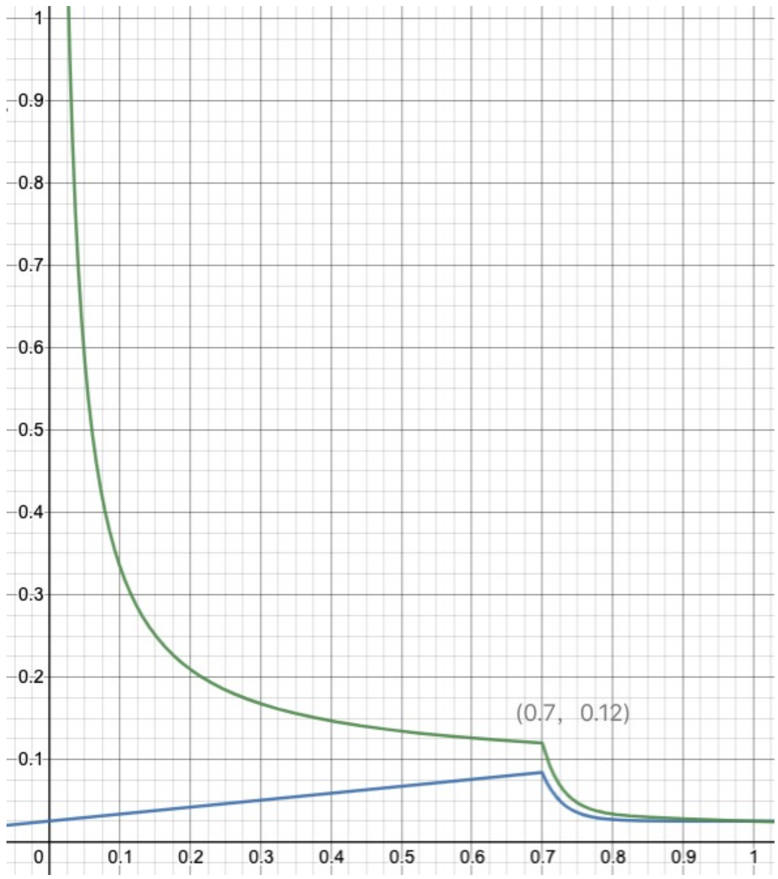
After a comprehensive evaluation, the algorithm formula parameters of the inflation rate and the rate of return in Ruby Network are as follows:

- $\chi_{ideal} = 0.7$
- $I_{ideal} = 0.12$
- $R_0 = 0.025$
- $d = 0.02$

Under this circumstance, the balance between the Staking rate and rate of return is as follows:

- When the Staking rate is lower than 70%, the average annualized rate of return of Staking will continue to increase, closing to 12%. Staking of more tokens will be encouraged.
- When the Staking rate is equal to 70%, the average annualized rate of return of Staking will be 12%;
- When the Staking rate is greater than 70%, and the average annualized rate of return of Staking is less than 12%, the Redemption instead of Staking will be encouraged.

The following figure shows the relationship between the simulated Ruby Network inflation rate, Staking rate, and annualized rate of return:



The X-coordinate is the Staking rate, the blue Y-coordinate is the annual inflation rate, and the green Y-coordinate is the annualized rate of return.

7.4 Block Reward

Every time a block is generated on the Ruby mainnet, the system will issue a certain amount of Ruby Tokens as block reward. The reward of each block will be determined by the real-time inflation rate of the network. The inflation rate, which is flexible, will be determined by the inflation formula. It will be affected by the actual Staking rate of the network.

Block rewards will be distributed in the following way:

- 80% for validators and nominators;
- 20% for the Ruby network Treasury, which will be used for community operation, project development, ecosystem incentives, etc.

7.5 Block Slash

When a validator misbehaves, such as off-line, double-signing, etc., it will be punished - their Staked Ruby will be fined for a certain amount alongwith their nominators.

The Ruby penalties will be affected by the status of the network. For example, if a high proportion of validators in the entire network is off-line or double-signing, the penalty for a single validator will be higher.

In extreme cases, for example, if over 1/3 of validators are double-signing, the Slash will be 100% of the Staked Ruby.

8 RubyDAO

Ruby Network will establish a community autonomous organization, namely RubyDAO. It will be based on Polkadot's on-chain governance mechanism or any future blockchain network we intend to deploy. Any Ruby token holder will be able to initiate governance proposals, adjust governance parameters, or apply for grants.

However, the RubyDAO governance level needs to be adapted to the development of the project. Especially in the early development stage when the product and business are still immature, the development will be hampered if the governance relies solely on the community autonomy. Market opportunities will also be missed. Therefore, the autonomy level of RubyDAO will continue to improve with the stage of development. Finally, the community will enjoy full autonomy.

Holders of Ruby tokens can make proposals for the following aspects in accordance with the rules:

1) System code upgrade;

2) Adjustment of system governance parameters, such as handling fee, amount of Staked Ruby, etc.;

3) Modification of governance rules;

4) Addition or modification of product features;

5) Financial support and incentive plan;

9 Treasury

In order to ensure the sustainable development and decentralization of the protocol, Ruby Network has set up a Treasury to incentivize individuals or teams who contribute to the project. Any use of the Treasury is approved through online governance. First, developers can submit their proposals to the community. Those voted and approved will be entitled to claim financial support from the Treasury.

The revenue of the treasury mainly come from the following two ways:

1) 20% of the Commission collected by RubyDAO from the transaction revenue of Data Marketplace.

2) 20% of the block reward.

10 Token Economics

10.1 RUBY Token

RUBY token is the native token of Ruby Protocol. The initial total supply is 1 billion. It will be used for the security of the Ruby Protocol chain, and the construction and incentives of the data transaction market.

In Ruby Network, the application scenarios of RUBY Token are as follows:

1) Scenario 1: When the Data Owner uploads data, he/she needs to pay RUBY token to obtain watermarking technical services and data storage services;

2) Scenario 2: Data Owner can use RUBY token for promotion, such as obtaining better system recommendations, launching Data Buyer review incentive activities, etc.;

3) Scenario 3: Data Buyer can use RUBY Token to deduct the data purchase fee from the Data Owner or the Data Collector;

4) Scenario 4: RUBY Token holders can delegate RUBY to Validators to obtain staking rewards;

5) Scenario 5: RUBY Token holders can participate in the governance voting of RubyDAO;

6) Scenario 6: When the Ruby product goes live, mining incentives will be launched. Users who upload data will receive RUBY Token rewards;

10.2 Token Distribution

Total supply of RUBY tokens is 1 billion, with the distribution plan as the following:

Token Allocation	**%**	**Vesting**
Seed Round	6.0%	Subject to 24-month vesting schedule with a 3-month cliff and quarterly vesting in 4–24 months
Private Round I	8.0%	5% on TGE, quarterly vesting for 18 months, starting one quarter after the listing
Private Round II	10.0%	10% on TGE, quarterly vesting for 12 months, starting one quarter after the listing
Public Sale	8.0%	25.0% on TGE, quarterly vesting for 6 months, starting one quarter after the listing
Parachain Bond Funding	15.0%	10% on TGE, monthly vesting for 48 months
Ecosystem Development	10.0%	10% on TGE, monthly vesting for 24 months
Foundation Reserve	15.0%	10% on TGE, monthly vesting for 24 months
Founding Team	10.0%	Subject to 48-month vesting from network launch, with a 12-month cliff and monthly vesting thereafter
Partners and Advisors	5.0%	Subject to 24-month vesting from network launch, with a 6-month cliff and monthly vesting thereafter

Token Allocation	%	Vesting
Developer Adoption Program	5.0%	Subject to vesting 2-year monthly linear vest. Unused tokens can be used for future sales
Early Staking Reward	4.0%	Subject to 2-year vesting schedule from network launch with a 6-month cliff and monthly vesting thereafter
Community Reward	4.0%	Subject to 2-year vesting schedule from network launch with a 6-month cliff and monthly vesting thereafter

10.3 Value Capture

The big scale of data transactions in the Ruby Data Marketplace will provide an important value capture scenario for RUBY token, acting as the core value support of RUBY token:

1) RUBY token is the fuel in the Ruby network. All related transactions on the Ruby chain will require RUBY token as commission;

2) In the data upload, storage, transaction, promotion, etc., of the Ruby Data Marketplace, RUBY tokens have real use values, which will greatly encourage users to hold them.

3) 70% of RubyDAO revenue will be used for repurchase and burning in the secondary market, which will reduce the inflation rate of RUBY tokens. The whole system may even become deflationary.

4) When submitting an on-chain proposal, community members need to stake a certain amount of RUBY tokens.

10.4 Community Engagement

- Bounty Program for General Community: We will reward users who contribute positively to community building and content creation through an Ambassador Program. The community management team will be available 24/7 to answer questions.
- Incentive Program for Data Monetization: After the main functions are completed, we will provide incentives for users to monetize their data on our platform. This is an encouragement for users to provide the data and purchase the data.
- Parachain Loan Offering Campaign: We may hold a Parachain Loan Offering and reward users for helping in our auction with Ruby Protocol tokens.
- Affiliated Program of Cryptographic Infrastructure: It is proven effective for user growth and can be integrated into Ruby's cryptographic infrastructure.

11 Roadmap

2020 Q3

- Project Establishment

2020 Q4

- Research on functional encryption

2021 Q1-Q2

- Official Whitepaper Release
- Official Website Launch
- Official Ruby community Launch

2021 Q3

- Functional encryption library V1.0 released
- Web 3.0 grant milestone 1 approved.

Q4 2021

- Micropayment Scheme and Relevant Substrate Module Released

Q1-Q2 2022

- Web 3.0 grant milestone 2 approved.

Q3 2022

- Substrate Builders Program application accepted
- Testnet V1.0 Launch
- Bug Bounty Program

Q4 2022

- Substrate Builders Program milestone 1 approved
- Mainnet V1.0 Launch

Q1 2023

- Access control for NFT-gated event V1.0 Launch

12 Development Milestones

Milestone 1 — Implement Cryptographic Modules

The main deliverable of this milestone includes:

- A cryptographic library that implements the inner product functional encryption and quadratic polynomial functional encryption.
- A substrate pallet that integrates the verification logic of the associated zero-knowledge proof for the legitimacy of the encrypted functional key.

Deliverable	**Specification**
License	Apache License 2.0
Documentation	We will provide both inline documentation of the code and a basic tutorial that explains how a user can (for example) spin up one of our Substrate nodes. Once the node is up, it will be possible to send test transactions that will show how the new functionality works.
Testing Guide	Core functions will be fully covered by unit tests to ensure functionality and robustness. In the guide, we will describe how to run these tests.

Deliverable	Specification
Article/Tutorial	We will publish a medium article that explains the functionality of the proposed cryptographic library and Substrate pallet delivered in this milestone.
Cryptographic modules	We will implement the cryptographic modules including inner product functional encryption and quadratic polynomial functional encryption [MSHBM2019] and the associated zero-knowledge proof. We will also implement the Substrate pallet that integrates the verification logic of the associated zero-knowledge proof for the legitimacy of the encrypted functional key.
Benchmark	Perform unit tests on the individual algorithms to ensure their safety benchmark on the gas cost and throughput of the proposed module.
Docker	We will provide a dockerfile to demonstrate the usage of our modules.

Milestone 2 — Client Implementation and Integration

The main deliverable of the milestone is the client that can trigger the aforementioned cryptographic modules and the micropayment scheme, and the necessary UI to enable the users to interact with all these algorithms.

Deliverable	Specification
License	Apache License 2.0
Documentation	We will provide both inline documentation of the code and a basic tutorial that explains how a user can (for example) spin up one of our Substrate nodes. Once the node is up, it will be possible to send test transactions that will show how the new functionality works.
Testing Guide	Core functions will be fully covered by unit tests to ensure functionality and robustness. In the guide, we will describe how to run these tests.
Article/Tutorial	We will publish a medium article that explains the functionality of the proposed client and UI delivered in this milestone.
Client modules	We will implement the client to support the key distribution and decryption of the functional encryption scheme [MSHBM2019]. The client will also generate the transaction that can trigger the aforementioned cryptographic modules and the micropayment scheme [MDJM2019], such as the encrypted functional key and zero-knowledge proof. We will provide a basic UI to take inputs from the users for all these algorithms and receive the outputs. More specifically, the UI will enable the data owner to input the raw data to generate the signed ciphertext and upload it to the cloud server. The UI will also allow the data purchaser to retrieve the functional key from the key authority and the ciphertext from the cloud and then perform the decryption. Finally, it will also allow these entities to interact with the Substrate module with the inputs and outputs defined in our architecture.
Benchmark	Perform unit tests on the individual algorithms to ensure their safety benchmark on the latency and usability of the proposed client functionalities.
Docker	We will provide a dockerfile to demonstrate the usage of our modules.

13 Substrate Builders Program Milestones

Milestone 1: Implementation of CP-ABE library and modification of zero-pool library

Implementation of ciphertext-policy attribute-based encryption (CP-ABE) library according to the standard on ABE proposed in ETSI tech specification. Different from our Web 3.0 grant delivery, the library will be full-fledged, and guarantee the following functionalities: The CP-ABE library will be chosen-ciphertext secure and support decentralized and multi authorities and attribute revocation. We will provide full documentation on how to use the CP-ABE library in practice. The existing zero-pool pallet for data monetization will be modified according to the new CP-ABE scheme. Since our project will focus on access control in our future development, we will adapt the data monetization part of our project to the access control requirement.

Milestone 2: Decentralized Setup of CP-ABE scheme

Provide a setup mechanism for the decentralized authorities. Add an API for a Ruby token holder to become an authority. This includes a substrate pallet for the holder to stake Ruby token before officially becoming a verified authority. A specification for the authority on how to validate the user's attributes. This specification will indicate how to translate standard access control policy into attribute universe, etc. A standard for converting an access-control policy into an attribute policy that can be used by the underlying encryption algorithms. Establish a secure and authenticated channel for the transfer of public parameters from the authority to the encryptor. An API for the encryptor to respond to the request for encryption.

Milestone 3: Revocation Mechanism of CP-ABE scheme

A standard for the attribute revocation mechanism. The standard will include the specification of the revocation list, and the time-based, and counter-based secret key revocation. An API for the encryptor to re-encrypt the ciphertext before revocation. An XCMP message layer for the notification mechanism between the encryptor and the event triggered by the update of the revocation list. A procedure for the authority to perform attribute revocation duty. This will include a backend algorithm for the authority to set up and maintain the attribute universe and attribute revocation list. It will also have an API for the authority to interact with the substrate pallet to update the revocation list. Launch an access control service based on Substrate pallet.

14 Future Plans

We will hire at least 8-10 more devs in the next three months. Meanwhile, we will apply for the Substrate Builder's Program. After that, Ruby Protocol wants to become a parachain for the

Polkadot network. We have some preparations for auction and we may design a community-wide LPO.

In phase 1, we will complete the implementation of cryptographic modules as a Substrate pallet that integrates the verification logic of the associated zero-knowledge proof for the legitimacy of the encrypted functional key.

In phase 2, our goal is to deliver the micropayment scheme and enable the users to interact with all these algorithms in a working product.

Finally, our goal is to provide an essential open API and SDK from a high-level perspective with the above tools, fully powering the private data management framework on Polkadot.

References

[GPSW06] Goyal, V., Pandey, O., Sahai, A., and Waters, B. (2006, October). Attribute-based encryption for fine-grained access control of encrypted data. In Proceedings of the 13th ACM conference on Computer and communications security (pp. 89-98).

[GGGJKLZ14] Goldwasser, S., Gordon, S. D., Goyal, V., Jain, A., Katz, J., Liu, F. H., ... and Zhou, H. S. (2014, May). Multi-input functional encryption. In Annual International Conference on the Theory and Applications of Cryptographic Techniques (pp. 578-602). Springer, Berlin, Heidelberg.

[GKPVZ13] Goldwasser, S., Kalai, Y., Popa, R. A., Vaikuntanathan, V., and Zeldovich, N. (2013, June). Reusable garbled circuits and succinct functional encryption. In Proceedings of the forty-fifth annual ACM symposium on Theory of computing (pp. 555-564).

[ALS2016] Agrawal, S., Libert, B., Stehle, D. (2016). Fully secure functional encryption for inner products, from standard assumptions. In Annual International Cryptology Conference. (pp. 333-362). Springer, Berlin, Heidelberg.

[B2017] Bourse, F. (2017). Functional encryption for inner-product evaluations (Doctoral dissertation).

[B2018] Buterin, V. (2018). On-chain scaling to potentially ~500 tx/sec through mass tx validation. Available at `https://ethresear.ch/t/on-chain-scaling-to-potentially-500-tx-sec-through-mass-tx-validation/3477`.

[BCTV2013] Ben-Sasson, E., Chiesa, A., Tromer E., and Virza M. (2014, August). Succinct non-interactive zero knowledge for a von Neumann architecture. In Proceedings of the 23rd USENIX Security Symposium, Security '14. Available at `http://eprint.iacr.org/2013/879`.

[BMEB2016] Bataineh, A. S., Mizouni, R., El Barachi, M., and Bentahar, J. (2016, June). Monetizing Personal Data: A Two-Sided Market Approach. *Procedia Computer Science*. **83** (pp. 472-479).

[CGW2015] Chen, J., Gay, R., and Wee, H. (2015, April). Improved dual system ABE in prime-order groups via predicate encodings. In Annual International Conference on the Theory and Applications of Cryptographic Techniques (pp. 595-624). Springer, Berlin, Heidelberg.

[FVBG17] Fisch, B., Vinayagamurthy, D., Boneh, D., and Gorbunov, S. (2017, October). Iron: functional encryption using Intel SGX. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (pp. 765-782).

[LCFS2017] Ligier, D., Carpov, S., Fontaine, C., and Sirdey, R. (2017, February). Privacy Preserving Data Classification using Inner-product Functional Encryption. In ICISSP (pp. 423-430).

[MDJM2019] Mehta, S., Dawande, M., Janakiraman, G., and Mookerjee, V. (2019, August). How to sell a dataset? pricing policies for data monetization. *SSRN Electronic Journal*.

[MSHBM2019] Marc, T., Stopar, M., Hartman, J., Bizjak, M., and Modic, J. (2019, September). Privacy-Enhanced Machine Learning with Functional Encryption. In European Symposium on Research in Computer Security (pp. 3-21). Springer, Cham.

[PHGR2013] Parno, B., Howell, J., Gentry, C., and Raykova, M. (2013, May). Pinocchio: Nearly practical verifiable computation. In 2013 IEEE Symposium on Security and Privacy (pp. 238-252). IEEE.

[RDGBP2019] Ryffel, T., Dufour-Sans, E., Gay, R., Bach, F., and Pointcheval, D. (2019). Partially encrypted machine learning using functional encryption. arXiv preprint arXiv:1905.10214.

[RRS2013] Reimsbach-Kounatze, C., Reynolds, T., and Stryszowski, P. (2013). Exploring the economics of personal data-a survey of methodologies for measuring monetary value.

[SC2017] Agrawal, S. and Chase, M. (2017). Simplifying design and analysis of complex predicate encryption schemes. In Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer, Cham.

[SGP2018] Sans, E.D., Gay, R. and Pointcheval, D. (2018). Reading in the dark: Classifying encrypted digits with functional encryption. IACR Cryptology ePrint Archive 2018, 206.

[TZLHJS2017] Tramer F., Zhang F., Lin H., Hubaux J.-P., Juels A., and Shi E. (2017) Sealed-glass proofs: Using transparent enclaves to prove and sell knowledge. In Security and Privacy (EuroSandP), 2017 IEEE European Symposium on, (pp. 19-34). IEEE.

[GPSW2006] Goyal, Vipul and Pandey, Omkant and Sahai, Amit and Waters, Brent. (2006) Attribute-based encryption for fine-grained access control of encrypted data Proceedings of the 13th ACM conference on Computer and communications security, (pp. 89-98).

[Mintlayer] Disrupting Asset Tokenization: an Introduction to Mintlayer's ACL https://www.mintlayer.org/news/2021-06-08-disrupting-asset-tokenization/

[litprotocol] Litprotocol https://litprotocol.com/

[nftGatedAccess] Another NFT Use Case: Access as Utility https://www.one37pm.com/nft/nft-access-passes-nftnyc

[pavemotors] pavemotors https://www.pavemotors.com/platform