v0.8.6 – LunarX –– 驱动DApp的全球通用...

LunarX–白皮书（中英文对照版）

（翻译：阿诚，微信：mobilecheng，原文请到标题下方的github地址下载）

LunarX: Universal Middleware for Data Driven Decentralized Applications, Part I (draft version 0.8.6)
(https://github.com/LunarX-ONE/White-Paper)

LunarX：驱动无中心应用的全球通用中间件，第1部分
（草案 0.8.6）
(https://github.com/LunarX-ONE/White-Paper)

Ben Fei, Email: feiben@lunarion.com
Jiangtao Li, Email: silver.lijt@gmail.com

Abstract—LunarX is an universal middleware, designed to build a decentralized, scalable, temper proof, anonymous and autonomous data management infrastructure for decentralized applications(DApp) and their users. It knits individual nodes together, serving upper layer applications with uniform data chain functionality, and every peer joining the LunarX network becomes a part of the whole data service system.

In real world businesses, not all the data is transactional that has to be put into a blockchain, but these great amount of data still requires properties of anonymous, autonomous, temper proof, scalability and traceability, in an untrusted peer to peer network. Some of these data is structured, other is unstructured. We propose an algebraic definition of commutative map to study these data structures within a uniform framework, and to see when given a data structure, which kind of encryption schemes are commutative with it, and how to construct a possible scheme. In order to manage these data structures and generate blocks, we developed DAG model for these purposes. The data model is highly localized, like a huge sparse but locally dense matrix covering the world. This is the reason why we also call LunarX a World Wide Table.

Built upon several key distributed computation and cryptography techniques, LunarX is designed for serving such data management purpose. It is a developing pilot project, hoping to contribute some ideas to the community.

摘要 – LunarX是一个支持数据驱动型DApp的全球分布中间件，旨在建立无中心、可扩展、防篡改、匿名自治的数据服务基础设施。LunarX将各个节点连接在一起，为上层应用程序提供统一的数据链功能。每一个加入LunarX网络的用户，都将成为这种数据服务的一部分，为世界上另一个角落的用户提供有价值的数据服务。

在现实世界的企业中，并非所有数据都是事务性的、必须要存储在区块链中，但这些数据仍然需要在不可信的P2P网络中具有匿名性、独立性、防篡改、可扩展性和可追溯性。其中一些数据是结构化的，另一些是非结构化的。我们定义了交换映射的代数概念，在一个统一的框架内研究这些数据结构，并且看看给定数据结构时，哪种加密方案与这种数据结构具有交换性，以及如何构建可能的加密方案。为了管理这些数据结构并生成块，我们为此开发了DAG模型。为了管理这些数据结构，LunarX的底层是一个全球覆盖但是局部稠密的稀疏矩阵，稠密的部分是一个DAG的模型来管理数据和生成Blocks。所以我们也把LunarX称为World Wide Table。

LunarX建立在几个关键分布式计算、密码学的基础上，专为实现这种数据管理目的而设计。这是一个正在开发中的试验性项目，希望能为社区贡献一些想法。

1. Introduction

Centralized online platforms provide convenient tools for users to access information and services from anywhere and anytime. These platforms normally manage big data centers, driven by variant database systems, from simple embedded k-v modules, to structured relational db systems, and new-sql big data clusters. But the greater concern in long term is privacy. These platforms not only have all the knowledge of their users' data, but also share users' identities and user's characteristics. Users have no way but to TRUST these platforms that their data and identities are safe. On the other hand, end users eventually pay for the high cost of these data centers.

1.介绍

中心化的在线平台为用户随时随地访问信息和服务提供了便利的工具。 这些平台通常管理由不同数据库系统驱动的大型数据中心，从简单的嵌入式k-v模块到结构化关系数据库系统，以及new-sql大数据集群。 但从长远来看，隐私是很大的问题。 这些平台不仅拥有用户的所有数据，而且还共享用户身份和用户特征。 用户没有办法，只能信任这些平台，相信自己的数据和身份是安全的。 另一方面，用户甚至还为这些数据中心的高成本买单。

In contrast to centralized platforms, decentralized applications(DApp for short) do not own users' data, users themselves do. cryptographic currency [1] and blockchain [2] technology solved the transaction part of data exchange. In fact, it is a data structure that chains blocks, and with each block, there are predefined fields used in recording properties of transactions, preventing "double spent" or other malicious actions on an account.

与中心化平台相比，分布式应用程序（简称DApp）并不拥有用户的数据，用户拥有自己的数据。 加密货币[1]和区块链[2]技术解决了数据交换的事务（交易）部分。 实际上，它是一个链块的数据结构，并且每个块中都有用于记录交易属性的预定义字段，可防止帐户上出现“双重花费”或其他恶意操作。

Transactional consistency is essential, but from a global perspective, only a little proportion of data is transactional. Other data has weaker requirement of globally consistency, but still requires a model with properties of secure, temper proof, anonymous, autonomous and traceability. It is a much more general and scalable data model for decentralized applications in describing the real world businesses.

事务一致性至关重要，但从全局角度来看，只有一小部分数据是事务性的。 其他数据对全局一致性的要求较弱，但仍需要具有安全性、防篡改、匿名性、独立性和可追溯性的模型。 这

是分布式应用程序在描述真实世界业务时更加通用和可扩展的数据模型。

This model is not as strict as transaction model. It is a distributed table that records users activities as rows, and does not collecte user data inside a data center, but distribute to possible peers joining in the network. Each individual user can only access his own data entries in the whole table, and these entries may be stored in multiple remote peers.

这个模型并不像交易模型那样严格。 它是一个分布式表格，表格的每一行记录用户的活动，不会在数据中心收集用户数据，而是分发给加入网络的某些节点。 每个用户只能在整个表中访问他自己的数据条目，这些条目可能存储在多个远程节点中。

In addition, columns need to be scalable(vertical scalability), since new services may be developed in the future. Although the total data may be thousands of Tera bytes, the overhead is balanced, since every participant, including the applications' providers will only store and compute the part that they have been authorized to access. The data model is highly localized, like a huge sparse but locally dense matrix covering the world.

另外，由于未来可能会开发新的服务，所以表格的列需要具有可扩展性（垂直可扩展性）。尽管表的总数据可能是数千TB，但开销是平衡的，因为包括应用程序提供者在内的每个参与者都将仅存储和计算他们有权访问的部分。数据模型是高度本地化的，就像遍布世界的一个巨大的稀疏但局部密集的矩阵。

This is the reason why we also call LunarX a World Wide Table. This distributed table design is a middleware that knits individual nodes together, serving upper layer applications with uniform data chain functionality.
There are several supporting techniques helping us to build the network: distributed hash table [8] [9], symmetric/asymmetric/homoporphic/non–homomorphic cryptography, consistent hash [11] [12], merkle tree/DAG [10] [27] [28], proof of storage and its variants, Reed– Solomon erasure coding [14], and some other peer to peer techniques.

N.Carmack
May 6, 2018

这就是为什么我们称LunarX为World Wide Table的原因。这种分布式表设计是一种将单个节点编织在一起的中间件，为上层应用程序提供统一的数据链功能。
有几种支撑技术可以帮助我们构建这个网络：分布式哈希表[8] [9]，对称/非对称/同态/非同态加密，哈希唯一性[11] [12]，merkle树/ DAG [10] [27] [28]，存储证明及其变体，Reed–Solomon抹除编码[14]以及其他一些P2P技术。

N.Carmack
2018年5月6日

2. Background

Databases collect data together, mapping them to structures for future update and analysis. Normally they are deployed in one or multiple centers, and machines, which run

db instances, trust each other. But in P2P network environment, since there is no trust and no stable service can be expected and provided, data security and availability become essential and critical.

2.背景

数据库保存数据，将它们映射成结构化以供将来更新和分析。 通常数据库被部署在一个或多个数据中心，运行数据库实例的机器互相信任。 但是在P2P网络环境中，由于没有信任，并且不能期望网络提供稳定的服务，所以数据的安全性和可用性变得至关重要。

2.1. Homomorphic Encryption

Homomorphic encryption is a form of encryption that allows computation on ciphertexts. Cloud computing platforms perform complex computations on homomorphically encrypted data, therefor clients sensitive data will not need to be exposed. Most recent advance on this field includes: The Brakerski–Gentry–Vaikuntanathan cryptosystem (BGV) [15], scale–invariant cryptosystem [17], the NTRU–based cryptosystem [18], and The Gentry–Sahai–Waters cryptosystem [19].

2.1 同态加密

同态加密是一种允许在密文上进行计算的加密形式。 云计算平台对同态加密的数据执行复杂的计算，因此客户端不需要暴露敏感数据。 该领域的最新进展包括：Brakerski–Gentry–Vaikuntanathan加密系统（BGV）[15]，体积不变加密系统[17]，基于NTRU的加密系统[18]和Gentry–Sahai–Waters加密系统[19]。

Fully homomorphic encryption schemes have weaker security properties than non homomorphic schemes. In our distributed data computing framework, we need to operate non–homomorphically encrypted data from remote, which is a much more strong constraint, leading to the algebra concept of commutative map(definition 5), but we redefine it for this context.

完全同态加密方案比非同态方案具有较弱的安全属性。 在我们的分布式数据计算框架中，我们需要从远程操作非同态加密的数据，这是一个更强的约束，导致了交换映射的代数概念（定义5），但是我们在这个上下文中重新定义了它。

2.2. BlockChain

In fact, a blockchain is a decentralized sequential transactional database. .....

But we should think again that do we really need everything to be a part of transactions? In decentralized applications, we need to record immutable facts, which part requires no globally consensus, but a consensus reached by all the participants who involved in the facts. And in some cases, we do allow double spent. For example, one publishes his article and shares it to his subscribers. In this scenario, this article has been ”spent” many times. What we have to do is recording these immutable activities, reaching consensus of all the subscribers and the author, and making sure they can not be tampered with.

2.2 区块链

实际上，区块链是一个分布式的顺序事务数据库。.....

但是我们应该再次思考，我们是否真的需要所有东西都成为事务的一部分？ 在分布式的应用程序中，我们需要记录那些不会变化的数据，这部分数据不需要全局共识，而是所有涉及到调用这些数据的参与者要达成共识。 在某些情况下，我们允许花费双倍代价。 例如，一个人发表他的文章并将其分享给他的订阅者。 在这种情况下，这篇文章已被“花费”了很多次。 我们要做的是记录这些不可变的活动，达成所有订阅者和作者的共识，并确保他们不会被篡改。

2.3. Proof of Storage

Since users and the service providers have no trust to each other, they both need to delegate someone else to verify the validity of the data. Most importantly, the delegated verifiers have no knowledge of users data. This is called publicly verifiable [3]. To users, it is also hard to judge whether or not the storage providers and verifiers collude in generating the proof for rewards. Even all the parties are honest, it is a much more unstable computational task. Peers may encounter hardware damage, or have their devices severed from the network. The complete data still needs to be reconstructed from the network during the time that some peers are out of service.

2.3 存储证明

由于用户和服务提供者彼此之间不相互信任，他们都需要委托其他人来验证数据的有效性。 最重要的是，委托的验证者不会知道用户的数据。 这被称为可公开验证[3]。 对于用户来说，很难判断存储提供者和验证者会不会为了利益，而在产生证明的过程中串通。 即使各方都是诚实的，这也是一个很不可靠的计算任务。 节点可能会遇到硬件损坏，或者从网络中断开了。 当一些节点不在服务时，仍然需要从网络重建完整的数据。

Existing projects on this domain are FileCoin [5], IPFS [4], Sia [6] and Storj [7]. IPFS defines a world wide content addressable file system with its innovative content-addressed block storage model and with content-addressed hyper links. It saves data in a Merkle DAG [27] (directed acyclic graph), which is versioned and similar to the architecture what the git version control system [28] is using. IPFS designed key components including identity management, networking, data exchanging and routing using established peer-to-peer protocols. In addition, it introduces a novel BitSwap protocol, inspired by BitTorrant [24] for the exchange of data blocks and relies on a Merkle DAG to manage its content-addressable storage. FileCoin lays on top of IPFS and builds an incentive system in rewarding disk space providers and auditors.

这个领域的现有项目有FileCoin [5]，IPFS [4]，Sia [6]和Storj [7]。 IPFS使用其创新的内容寻址块存储模型和内容寻址超链接，创建了一个全球范围的内容寻址文件系统。 它将数据保存在Merkle DAG [27]（有向无环图）中，是版本化的，与git版本控制系统[28]使用的架构类似。 IPFS使用已完成的P2P协议设计了关键组件，包括身份管理、联网，数据交换和路由。 此外，它还引入了一种新颖的BitSwap协议，此协议受BitTorrant [24]的启发，用于交换数据块，并依靠Merkle DAG来管理其可内容寻址的数据存储。 FileCoin在IPFS的基础之上建立了一个激励体系，存储空间提供者和审核者可从FileCoin网络中获得奖励。

The supporting component of these systems is proof of storage. Filecoin invented proof of Replication and proof of spacetime [25] for preventing Sybil attacks, Generation attacks and Outsourcing attacks. If they steal the replica from other nodes, the attack is called outsourcing attack. Or, if they somehow can generate a replica, or something else that can be sufficient in proof, the attack is called generation attack. But these attacks are scheme-specific, meaning that if we adopt data replication as our storage model, then malicious nodes will probably generate the proof of a replica on demand but actually they do not have the data in their local storage. LunarX may encounter some other attacks, which will be addressed after we define the data model and architecture in the next section.

支撑这些系统运转的机制是：存储证明。 Filecoin发明了复制证明和时空证明[25]用于防止女巫攻击、生成攻击和外包攻击。 从其他节点窃取复制品称为外包攻击。 以某种方式生成副本或可以证明能生成足够的信息，此攻击称为生成攻击。 但是这些攻击是针对特定方案的，这意味着如果我们采用数据复制作为我们的存储模型，那么恶意节点可能会根据需求生成复制证明，但实际上它们没有将数据存储在本地存储中。 LunarX可能会遇到其他一些攻击，我们将在下一节定义数据模型和体系结构后解决这些攻击。

3. Definitions and Data Model

Outsource computational tasks from a thin client with relatively weak computational device to a more powerful computation services (worker) risks the leak of information. In this section, we formalize the concepts that support information confidentiality, and in the next section we focus on computational verification.

3.定义和数据模型

将计算任务从具有相对较弱计算能力的瘦客户端外包给更强大的计算服务（工作站）可能会泄漏信息。 在本节中，我们将具象化支撑信息保密性的概念，接下来的部分我们将重点放在计算验证上。

3.1. Commutative Map

Definition 1. A map is an operation that transforms a data set D to another structure without loss of information. m:D → D (1).

We define:

Definition 2. A unit map I is an operation that keeps the data structure unchanged.

Example 1. The following examples do not change the data structure, hence are unit maps:

replication of one data set.

query certain value from a data set.

Definition 3. if a∘b = I, we call a is the inverse map of b, and vice versa.

For example, encryption enc(.) is the inverse map of decryption dec(.).

3.1 交换映射

定义1. 映射是一种转换数据的操作，将D映射为另一个结构而不丢失信息。

m：D→D (1)

我们定义：
定义2. 一个单元映射 I 是保持数据结构不变的操作。
示例1. 以下示例不会更改数据结构，因此是单元映射：
复制一个数据集。
查询数据集中的某个值。
定义3. 如果 a ◦ b = I，我们称a为b的逆映射，反之亦然。
例如，加密enc(.) 是解密dec(.) 的逆映射。

All maps, with the join ◦ of any two maps, form a semi-group M. Let m1, m2 ∈ M, d ∈ D, the properties of associativity and identity are satisfied:
m1 ◦ (m2(d)) = (m1 ◦ m2)(d) (2)
I◦(d)) = d (3)

Since M is not a ring, and D is not an abelian group, D with an action of M does not form a M — module. Hence there is no distributivity(or bilinearity) of this structure.
Definition 4. A projection prj is a mapping of a data structure into a subset.

Projection may lose information, but will extract some meaningful results out of the whole original data set. Mathematical functions in data statistics and analysis are normally projections, like: multiplication, addition, summing, etc. Set operations like union and intersection are projections. And so are their combinations.

加入任意两个映射的所有映射形成一个半群M。 令m1，m2∈M，d∈D，关联性和同一性的性质得到满足：
m1◦（m2（d））=（m1◦m2）（d） （2）I◦（d））= d (3)
由于M不是一个环，并且D不是一个交换群，所以具有M行为的D不构成一个M模块。 因此，这种结构没有分配（或双线性）。
定义4. 一个prj投影是数据结构到子集的映射。
投影可能会丢失信息，但会从整个原始数据集中提取一些有意义的结果。 数据统计和分析中的数学函数通常是投影，如：乘法，加法，求和等。集合运算如联合和交叉是投影，他们的组合也是如此。

Definition 5. We define that two maps m1 , m2 ∈ M are **commutative** if m1 ◦m2 =m2 ◦m1.
Apparently, only those maps that are commutative with
encryption, can be distributed to untrusted peers.
Lemma 1. Every map is commutative with the unit map I: I◦m=m◦I (4)
Proof 1. Obvious.
Example 2. Homomorphic Encryption: Some encryption schemes are homomorphic with respect to their specific operations. In our context, these operations are commutative with certain encryptions. For example, if an encryption algorithm h enc(.) has homomorphic properties for string concatenation cat(.), which is a projection of a string space, then one can concatenate the ciphertext:
s = cat(h_enc(Str1), h_enc(Str2)) (5)
and
h_dec(s) = cat(Str1, Str2) (6)

where h_dec(s) is the inverse map of h_enc(.). Apparently, it is commutative:
cat(.)◦h enc(Str1,Str1)=h enc(.)◦cat(Str1,Str2) (7)

.
定义5. 我们定义两个可变化的映射m1，m2∈M。如果 m1 ◦m2 =m2 ◦m1.，
显然，只有那些可加密交互的映射，可以分发给不受信任的节点。

引理1. 每个映射都与单元映射 I 交换： I: I◦m=m◦I (4)
证明1. 明显。
例2.同态加密：一些加密方案就其具体操作而言是同态的。 在我们的上下文中，这些操作与某些加密是可交换的。 例如，如果一个加密算法henc(.)具有字符串连接cat(.)的同态属性（字符串空间的投影），则可以连接密文：
s = cat(h_enc(Str1), h_enc(Str2)) (5)
和
h_dec(s) = cat(Str1, Str2) (6)

其中 h_dec(s)是h_enc(.)的逆映射。 显然，它是可交换的：
cat(.)◦ h_enc(Str1, Str1) = h_enc(.)◦ cat(Str1, Str2) (7)

The math concept defined in this section has fruitful intrinsic structures to be discovered. As we mentioned above, it is not a M − module, then what it is exactly? Since this is a system design article, we will not go deep on this subject, but will draft another paper to elaborate this theory.

本节中定义的数学概念具有丰富的内在结构有待研究。 正如我们上面提到的那样，它不是一个M模块，但它究竟是什么？ 由于本文是系统设计白皮书，我们不会深入研究这个主题，但会起草另一篇文章来阐述这一理论。

3.2. Typical DB Structures

Online data-related services usually depend on the basic data access capability provided by the centralized database systems. Starting from analyzing typical data structures, we will discuss the design of an universal middleware.

3.2 典型的数据库结构

各种在线的数据相关服务，通常依赖中心化数据库系统提供的基础数据访问功能。从分析典型数据结构开始，我们将讨论通用中间件的设计。

3.2.1. Numerical Calculations. Numerical calculations are projections mapping data from a high dimensional space to a discrete space. For example:
addition of two numbers is not commutative with non- homomorphic encryption. But addition may be commutative with homomorphic encryption schemes.
Ordering of a random number sequence is not commutative with encryption.

3.2.1 数值计算。 数值计算是将数据从高维空间映射到离散空间的投影。 例如：
两个数字的加法与非同态加密不可交换。 但是加法可能与同态加密方案是可交换的。
随机数字序列的排序与加密不可交换。

There are many data structures invented in the past decades for various data management purposes. We here discusses some typical structures that are mostly used in database implementations. Both relational databases and new-sql column

based big data systems may have different indexes on columns in order to be able to efficiently find rows that match a particular query. Indexes are auxiliary data structures that represents the information in different ways. Hence they are maps between data structures. For numerical columns, B+ tree is a standard choice for range queries. And on string or text columns, inverted index and hash table are normally used to perform keyword search in constant complexity.

出于各种数据管理目的，在过去的几十年中发明了许多数据结构。 我们在这里讨论一些主要用于数据库实现的典型结构。 为了能够有效地查找与特定查询相匹配的行，关系型数据库和基于new-sql列的大数据系统在列上可能具有不同的索引。 索引是以不同方式表示信息的辅助数据结构。 因此它们是数据结构之间的映射。 对于数字列，B+树是范围查询的标准选择。 在字符串或文本列上，倒排索引和哈希表通常用于执行恒定复杂度的关键字搜索。

3.2.2. B tree. B tree and its variant B+ tree [26] are data structures that maintain the order of numerical random data sequences, and perform searches, insertions and deletions in logarithmic time. The process building a B+ tree is a map that attaches each element to a path from the root of the tree. The leaf nodes of the tree are ordered, so that we can find any range of data without going through all the data set. It is quite efficient when the data set grows huge.

3.2.2 B树。 B树及其变体B+树[26]是一种数据结构：维护随机数据序列的顺序，并在一定时间内执行数据的搜索、插入和删除。 构建B+树的过程是将每个元素附加到树根的路径的映射。 树的叶节点是有序的，这样我们就可以在不遍历所有数据集的情况下找到任何范围的数据。 当数据集增长巨大时，它非常有效。

But if encryption schemes can not preserve the order of the original data, applying B+ tree upon an encrypted data set holds no meaning. Order-preserving encryption scheme(OPES for short [20]) enables to apply comparison operation directly on encrypted data. But this advantage is accompanied by a weakness of the system. An attacker does not have to determine the exact sensitive data d, on the contrary, it is still a successful attack if he can estimate a tight enough interval including d with a high enough probabilistic confidence level, say, 95%.

但是，如果加密方案不能保留原始数据的顺序，那么在加密数据集上应用B+树就没有意义。 保序加密方案（简称OPES [20]）能够直接对加密数据进行对照操作。 但这种优势伴随着系统的弱点。 一个攻击者不必确定确切的敏感数据d，如果他能够估算足够紧密的时间间隔，包括具有足够高的概率置信水平（例如95%）的d，他就能发起成功的攻击。

An attacker can repeatedly generate random numbers to test the encryption system and get the encrypted data, by comparing with the users' encrypted data, he is able to estimate what the original plain data is. Hence the domain of users' data must keep secrete.

攻击者可以重复生成随机数来测试加密系统并获得加密数据，通过与用户的加密数据进行比较，他可以估算原始数据是什么。 因此用户数据域必须保持安全。

Monotonic functions are order preserving, so are their composites. Here we construct an example function, which is piece-wise linear with randomly generated scaling parameters. Users should construct their own in practical applications.

单调函数是保持顺序的，它们的复合也是顺序的。 这里我们构造一个示例函数，它是随机生成的可变参数的分段线性函数。 用户应该在实际应用中构建自己的。

Example 3. let L = {l0,l1,l2,...,ln}, where li ∈ (0,∞) are randomly generated.
Define [0,1] the domain of f(x), and randomly split it into n intervals: [xi, xi+1), i ∈ {0, 1, ..., n − 1}:

$$f(x) = \begin{cases} 0 & x = 0, \\ \Sigma_{j=0}^{i} l_j + \frac{l_{i+1}}{x_{i+1}-x_i}(x - x_i) & x_i \le x < x_{i+1}. \end{cases}$$

Insert and Query:
1) For a randomly incoming data sequence D = {d1, d2, ...}, user select a locally monotonic function f with domain [x1,x2], calculate f(di).

例3. 设L = {l0，l1，l2，...，ln}，其中li∈（0，∞）是随机产生的。
定义[0,1] f（x）的域，并将其随机分为n个区间：[xi，xi + 1），i∈{0,1，...，n−1}：
（原文数学公式如上）
插入和查询：

1. 对于一个随机输入的数据序列D = {d1，d2，...}，用户选择一个局部单调函数f，域为[x1，x2]，计算f（di）。

2) Transforms di linearly into the monotonic domain of f. Since the rang of di is unknown, we may multiply a sufficient small factor ε to the data set to make sure all the possible values fall into [x1,x2]. ε is application specific. Or we can choose a function with monotone domain from −∞ to ∞.
3) Sends f(di) and the signature to the remote peer.
4) On remote peer, verify signature firstly, if fails, reject the request.
5) Remote peer inserts into its local B+ tree. Remote peer has no knowledge of the function the user construct, he will not possible to reconstruct the function.
6) In any range query from xi to xj, user send [f (xi), f (xj)] to the remote peer and get the list of result points. Use the inverse function of f to get the original ranged data set.

2）将线性变换成f的单调域。 由于di的范围是未知的，我们可以将足够小的因子ε乘以数据集以确保所有可能的值落入[x1，x2]。 ε是特定应用程序。 或者我们可以选择一个单调域从−∞到∞的函数。
3）将f（di）和签名发送给远程节点。
4）在远程节点，首先验证签名，如果失败，拒绝请求。
5）远程节点插入到其本地B+树中。 远程节点不了解用户构造的函数，他不可能重构此函数。
6）在从xi到xj的任何范围的查询中，用户将[f（xi），f（xj）]发送到远程节点并获取结果点列表。 使用f的反函数来获取原始范围的数据集。

3.2.3. Hash Table. The complexity of accessing data by hash table is O(1). Put(D) stores data D at the array index computed by hash function hash(.), and get(D) fetches D from the index. So actually get(D) = query(.) ◦ hash(D), where query(.) gets data from the index.

Lemma 2. get(.) is commutative with encryption enc(.):
Proof 2. dec(.)◦query(.)◦hash(.)◦enc(.) = query(.)◦hash(.) (8)
since enc(.) is the inverse element of dec(.), we left multiply enc(.) on both sides and get:
query(.)◦hash(.)◦enc(.) = enc(.)◦query(.)◦hash(.) (9)

3.2.3 哈希表。 哈希表以O（1）的复杂度访问数据。 Put(D)方法将数据D存储在由哈希函数hash(.)计算的索引数组处，get(D) 方法从索引中取出D。所以 get(D) = query(.)◦hash(D)，其中query(.)从索引获取数据。

引理2. get(.) 与加密enc(.)方法是可交换的
证明2： dec(.)◦query(.)◦hash(.)◦enc(.) = query(.)◦hash(.) (8)

因为enc(.)是dec(.)的逆操作，所以我们在等号两边都放置enc(.)，得到：
query(.)◦hash(.)◦enc(.) = enc(.)◦query(.)◦hash(.) (9)

3.2.4. Inverted Index. Actually an inverted index is a hash table with each entry attached a list of content ids(or addresses). It is popular and wide adopted by large scale search engines [22]. When in query, one can quickly locate where the keywords are, and fetch all the content including them by visiting the storage in constant complexity. Hence the following lemma is apparent.

3.2.4 倒排索引。 实际上，倒排索引是一个哈希表，每个条目都附有一个内容ID（或地址）列表。 它受到大型搜索引擎的欢迎和广泛采用[22]。 在查询时，可以快速定位关键字的位置，以恒定的复杂度访问存储器来获取所有内容。 因此下面的论点是显而易见的。

Lemma 3. Operations, including union, intersection and any combination of the two, on this data structure is commutative with encryption.
union ◦ enc = enc ◦ union (10)
intersection ◦ enc = enc ◦ intersection (11)

引理3. 在这个数据结构上的操作，包括联合，交集以及两者的任意组合，都可以与加密交换。
union ◦ enc = enc ◦ union (10)
intersection ◦ enc = enc ◦ intersection (11)

3.3. Data Model
Multiple version control: a git-like DAG model for multiple versions of a data entry(figure 1).

Tamper resistance: to be done.
As we mentioned before, The data model is highly localized, like a huge sparse but locally dense matrix covering the world. to be more precisely, this locally dense part is a DAG (directed acyclic graph), which is not only used for version tracking, but also used to meet many other requirements of structured data processing.
We will keep updating data model during project developing.

3.3 数据模型
多版本控制：一种类似git的DAG模型来进行数据条目的多版本控制（图1）。

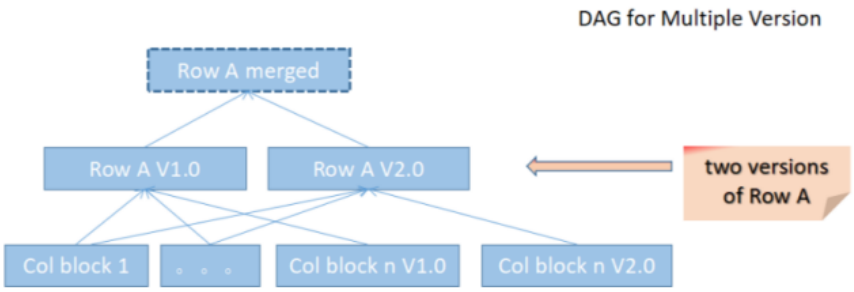

Figure 1. Multiple Version Control

防篡改：（需要完成）
如前所述，数据模型是局部高度稠密的，就像覆盖全球的巨大稀疏但局部密集的矩阵。更准确地说，这种局部稠密部分是DAG (有向无环图)，它不仅用于版本跟踪，而且还用于满足结构化数据处理的许多其他要求。
我们将在项目开发期间不断更新数据模型。

4. Network Definition, Attacks and Solutions

4.1. Properties

A scheme of proving something(not only storage), in an environment absent trustiness of each other, must has such properties:
P1) Transparent: no prover is able to generate a valid proof if it has no original data.
P2) Zero knowledge: a verifier must validate a proof without knowing the original data. Verifier can be anyone, and this is called publicly verifiable [3].
P3) No conspiracy: a verifier has no chance to collude with any prover, generating positive confirmation of fake storage, or negative proof of an honest service provider.

4.网络定义，攻击和解决方案

4.1 属性

在缺乏彼此信任的环境中，证明某些东西（不仅是存储）的方案必须具有这样的属性：
P1）透明：如果证明者没有原始数据，证明者不能生成有效证明。
P2）零知识：审核者必须在不知道原始数据的情况下验证证明。 审核者可以是任何一个人，这被称为可公开验证[3]。
P3）没有阴谋：审核人没有机会与任何被证明人勾结，避免对假存储做出正面确认，或者对真正的服务提供者做出负面证明。

P4) Distinguishable identities: Identify every peer that participates in the network, ensuring that for a certain data piece, different peer has different signature, hence stops outsourcing attack [25].

P5) Robustness: in a globally distributed system, peers may often be disconnected from the network, no responding to any challenger or data request. Obviously this is not malicious. After it is online again, auditors should be notified and begin the data possession validation again. So a criteria to distinguish malicious actions or just normal disconnection is quite necessary.

P4）身份确认：确定参与网络的每个节点，确保对于某个数据片段，不同节点具有不同的签名，从而阻止外包攻击[25]。

P5）健壮性：在全球化的分布式系统中，某个节点可能经常与网络断开连接，不响应任何访问者或数据请求。 显然这不是恶意的。 这个节点再次上线后，审核员应该会收到通知并开始这个节点数据所有权的验证。 因此，如何区分恶意操作或是正常的断开连接，这个机制非常必要。

P6) Retrievability: proof of retrievability [29] [30](POR for short) is a scheme to make sure a service provider has no chance to blackmail users. In one hand, a provider can prove he has the data belonging to a user, but in the other hand, alters the code so that he will not release the data to the owner unless the owner pay something to him. PORs hence enable the data owner to challenge the provider several times, and at each time gets one piece of data along with the proof. After a round of challenges, the owner can reconstruct the complete data. POR is an enhancement of PODP(Proof of Data Possession) [23] [31]. But according to P2), any verifier must not has the ability to reconstruct data by repeatedly challenging a provider. So in LunarX, we require that users have to encrypt their original data before submitting them to the network.

P6）可检索性：可检索性证明[29] [30]（简称POR）是一个方案，用来确保服务提供者没有机会勒索用户。 一方面，服务提供者可以证明他拥有用户的数据，但另一方面，修改代码以便他不会将数据发布给用户，除非用户给他支付了什么。 因此，POR使数据拥有者可以多次请求提供者，并且每次都会获取一份数据以及证明。 经过一轮请求后，数据拥有者可以重建完整的数据。 POR是PODP（数据拥有证明）的增强[23] [31]。 但根据 P2），任何审核者都不能重复请求数据提供者来重建数据。 因此，在LunarX中，我们要求用户在提交数据给网络之前必须加密原始数据。

P7) Memoryless: if a peer proves its storage of a data D in time slot t, its probability p(t + s) of possessing the data for extra time slot s is the same to p(s). This motivates us to use a exponential distribution to generate challenges to peers. The promising point is that when the time slot gets denser, the cost of attack grows exponentially, due to the advantage of exponential distribution.

P7）无记忆：如果一个节点在一个时间间隔（时隙） t 中证明它存储了一个数据D，那么它拥有额外时隙 s 的数据的概率 p(t + s) 与 p(s) 相同。 这激励我们使用指数分布来为节点产生请求。 有希望的一点是，当时隙变得更密集时，由于指数分布的优势，攻击的成本呈指数增长。

4.2. Roles

The network consists of three major roles: DApp and their users, service provider, tracker. The following scenarios elaborate how these roles cooperate with each other in the LunarX network.

DApps and their users:
DApps run on LunarX and their users who subscribe the services they published. Both sides are data senders, and the receiver is no longer a private database, but the LunarX open network. For example, a storage business provides a cloud-storage application, its

clients may choose to submit files to LunarX, and the business has no way to access these files. Only thing it possibly have is the data that clients authorized it to have.

4.2 角色

LunarX网络由三个主要角色组成：DApp和其用户，服务提供者，追踪者。 以下场景阐述了这些角色如何在LunarX网络中彼此协作。

DApps及其用户：

DApps运行在LunarX网络上，DApps发布各种服务，用户订阅这些服务。 双方都是数据发送者，接收者不再是私有数据库，而是LunarX的开放网络。 例如，存储服务商提供云存储应用程序，客户可以选择向LunarX提交文件，而服务商无法访问这些文件。获取数据的唯一可能是客户授权这些数据。

Service Providers:

Any user can join the LunarX network to be a data service provider, selling his spare disk space to other peers in the network. The LunarX daemon automatically accepts records from the network and insert them into local table shard. At regular intervals, the service provider earns incentive tokens for his service, as long as automatic proofs show that the data structure is still available and complete, without being altered or damaged. LunarX employs an incentive blockchain system to mine tokens. During the repeatedly proofing of valid data computing, blocks will be created, and with them a certain amount of tokens as the mining reward will be sent to the provider's wallet address. Also, the blockchain manages exchanges of tokens and services.
Not only local file systems can be of service, the distributed file systems (DFS) are also good candidates for professional users. There are many industrial level DFS implementations, for example the hadoop distributed file system(HDFS) [32]. Typically, this option is for traditional data center operators.

服务提供者：

任何用户都可以加入LunarX网络成为数据服务提供者，将其多余磁盘空间出售给网络中的其他节点。 LunarX守护进程自动接受来自网络的记录并将它们插入到本地表分片中。 服务提供者提供服务，只要自动校验显示数据结构可用且完整、数据没有被更改或损坏，在固定周期内就可以获得代币奖励。
LunarX采用区块链奖励系统来挖矿。 在有效数据计算的重复证明过程中，将创建区块，并产生一定数量的代币，作为挖矿奖励发送到服务提供者的钱包地址。 区块链也管理代币交易和各种服务。
不仅个人电脑的本地文件系统可以提供服务，分布式文件系统（DFS）也是专业用户的理想选择。 有许多工业级别的DFS实现，例如hadoop分布式文件系统（HDFS）[32]。 通常，这些适用于传统数据中心运营商。

Trackers:

Trackers validate transactions between service users and service providers. In addition, each of them drives a bunch of verifiers to audit data service providers of the retrievability of the data and availability of the services they deployed. And trackers will be rewarded by tokens for their service as well.

追踪者：

追踪者验证使用服务的用户和服务提供者之间的交易。 另外，每一个追踪者驱动一批审核者来审核数据服务提供者的数据可检索性以及他们部署的服务的可用性。 追踪者提供自己的服

务而获取代币奖励。

4.3. Rules

The following rules are common sense, but we still list them here for the completeness of this work. Readers shall keep these in mind in the following discussion to avoid possible logical fallacies.

1)Service providers have no original data. What they are permitted to have is the encrypted data only.

2)Verifier has zero knowledge of the original data, neither it can reconstruct the original data via repeatedly challenging. Although we expect the auditors to be honest, the protocols of verification/audition are open. Then any peer has the chance to imitate a verifier to steal data.

4.3 规则

以下规则是常识，但为了这项工作的完整性，我们仍然在这里列出它们。 读者应在下面的讨论中记住这些以避免可能的逻辑谬误。

1）服务提供者没有原始数据，拥有的只是加密数据。

2）审核者对原始数据没有任何的了解，也不能通过重复请求数据来重建原始数据。 虽然我们期望审核者是诚实的，但验证/审核的协议是开放的。 然后任何节点都有机会模仿审核者窃取数据。

3)Most of the possible attacks or cheats we talk about have a prerequisite that the attackers have the ability to tamper with the code and disguise their nodes as normal nodes in the network, following all the LunarX protocols in communication, but with fabricated data encapsulated in the messages for malicious purposes.

According to the above rules, together with the properties in section 4.1, we may be able to design our defence system against attacks. We firstly begin to discuss the possible attacks in an untrusted network and how our defence works.

3）我们所谈论的大多数可能的攻击或欺骗有一个先决条件，即攻击者有能力篡改代码并将其节点伪装成网络中的正常节点，遵循通信中的所有LunarX协议，但是在消息中封装了伪造数据用于恶意目的。

根据上述规则，连同4.1节的属性，我们可以设计防御系统来抵御攻击。 我们首先开始讨论在不可信网络中可能发生的攻击以及我们的防御是如何工作的。

4.4. Attacks/Flaws

From now on, we use data D to represent a collection of table entries, and enc(D) is the encrypted version of the data by symmetric or asymmetric cryptographic methods. Normally, some entries may serialize very big binary or text data, We will not address how a big entries being segmented to relatively small pieces, since it is straight forward. What we will focus is how enc(D) is going to be saved(in merkle tree or DAG), secured, verified, and reconstructed. Since if every piece of a enc(D) is safe and retrievable, the whole enc(D) is safe and retrievable.

4.4 攻击/缺陷

从现在开始，我们使用数据D来表示表条目的集合，使用enc(D)来表示数据D的加密版本（用对称或非对称加密方法来加密）。通常情况下，一些条目可能会序列化成非常大的二进制或文

本数据，因为它很简单，所以我们不会考虑大条目如何分割为相对较小条目的问题。 我们将关注的是enc(D)将如何被保存（在merkle树或DAG中）、安全、验证和重建。 因为如果每一部分enc(D)都是安全可检索的，那么整个enc(D)就是安全可检索的。

Sybil attack [33] [34]: Sybil identities S1, S2, ..., Sn, controlled by an attacker, claim they all have a copy of enc(D), and generate valid proofs when challenges come from verifiers. But actually, they only have one(or significantly lesser) copy of enc(D).

女巫攻击[33] [34]：攻击者控制的多个女巫身份S1，S2，...，Sn，声称他们都拥有enc(D)的副本，并在审核者验证数据时生成有效证据。 但实际上，他们只有一个（或更少）的enc(D)副本。

Natural Damage: In a globally distributed environment, services provided by nodes from everywhere is unstable. They may have disk errors in running, network jam in communicationor whatever problems of devices that they can not response any request in time. So for the robustness of the system, we leverage Reed-Solomon erasure coding [13] [14] for recovering data from error nodes. That is to say, if some of the nodes fail to respond data request, we still can recover the complete data from the remaining nodes. In our test, Reed-Soloman coding reaches 400MB/G/core on our desktop computer.

自然灾害：在全球化的分布式环境中，各处节点提供的服务是不稳定的。 它们可能在运行时出现磁盘故障、通信中出现网络堵塞，设备可能出现任何问题而无法及时响应一切请求。 因此，为了系统的健壮性，我们利用Reed-Solomon抹除编码技术[13] [14]从错误节点恢复数据。 也就是说，如果某些节点未能响应数据请求，我们仍然可以从其余节点恢复完整的数据。 在我们的测试中，Reed-Soloman编码在我们的台式电脑上达到了400MB / G /内核。

Forging Attack: Provider does not honestly store enc(D), but generates proof on demand, i.e. when it receives a challenge, it steals the data from another peer. If a scheme relies on multiple replication of the original data, a fake storage could be forms of outsourcing or generation just in time [5]. Since LunarX has no duplication of the original data in any untrusted peers, attackers then have nowhere to steal the data in responding challenges. This attack has no chance to perform.

伪造攻击：服务提供者虚假存储 enc(D)，但是按照要求产生证明，即当它接收到请求时，它偷走来自另一个节点的数据。 如果一个方案依赖于原始数据的多次复制，那么假存储会很快形成外包或生成要求[5]。 由于LunarX不拥有不信任节点的原始数据的副本，因此攻击者无从窃取数据。 这些攻击是没有机会执行的。

Instead, a possible cheat from a provider is in this way: provider gets the data and its merkle tree, but he only persists the merkle tree and discards the data. Hence he can claim more space to auditors for more rewards. We defend this by randomly injecting water print in the encrypted data, and we name it proof of random injection(PORI):

相反，服务提供者可能会作弊的方式是：提供者获取数据及其merkle树，但他只保留merkle树并丢弃数据。 因此，他可以向审核人员证明自己有更多的存储空间以获得更多奖励。 我们通过在加密数据中随机注入水印来防止这一点，我们将其命名为随机注入证明（PORI）：

Setup:

1) User U submits E = enc(D) to LunarX.
2) LunarX segments E into pieces Ei , i = 1, 2, 3, ..., n, constructs Reed-Soloman data blocks RSj (E), j = 1,2,3,...,m, where m > n, depending on what code rate we choose.
3) Injects in random positions pk (RSj), k ∈ Z in some of these pieces RSj with special water print ck(RSj).
4) LunarX tells U these ck(RSj) and pk(RSj).

Challenge:
5) Verifier issues a challenge with pk(RSj) for RSj, k and j ∈ {1, 2, ..., n} are randomly selected.
6) Prover, i.e. the provider, answers this challenge by seek in RSj and read the water print cj,k(P).

Verify:
7) For all j and k, Verifier receives the code cj,k(P).
8) Compare cj,k(P) and ck(RSj), equals then the proof is valid.

设置：
1）用户U向LunarX提交 E = enc(D)。
2）LunarX将E分割成Ei，i = 1,2,3，...，n，构成Reed-Soloman数据块RSj（E），j = 1,2,3，...，m，其中 m > n，这取决于我们选择的码率。
3）把特殊的水印ck（RSj）注入到RSj一些片段中的随机位置pk（RSj），k∈Z。
4）LunarX告诉U这些ck（RSj）和pk（RSj）。

质询：
5）审核者用pk（RSj）对RSj发出请求，k和j∈{1,2，...，n}是随机选择的。
6）证明者，即提供者，通过在RSj中寻找并读取水印cj，k（P）来响应这个请求。

校验：
7）对于所有j和k，审核者接收代码cj，k（P）。
8）比较cj，k（P）和ck（RSj），如果相等那么证明就是有效的。

Hostage Attack: A provider holds the data piece but does not release it to its owner(the user who submit the data). The retrievability (P6, section 4.1) prevents this attack.

人质攻击：服务提供者拥有部分数据，但没有将其释放给所有者（提交数据的用户）。 可获取性（第6页，第4.1节）阻止了这种攻击。

Conspiracy Attack: Verifiers and providers collude. A provider offers no service but collude with a bunch of verifiers for proofing the integrity and retrievability of the data he does not really has. This is the why the rule 2 in section 4.3 claims that a verifier can not have any piece of the original data. (to be more specific)

阴谋攻击：验证者和提供者串通。 提供者不提供任何服务，但与一群验证人串通，证明他没有真正拥有数据的完整性和可检索性。 这就是为什么4.3节中的规则2声明验证者不能拥有任何原始数据。 （更具体）

Ciphertext only Attack [20] [21]: An attacker only has access to all the encrypted data, but known nothing about the domain of the original data.

仅密文攻击[20] [21]：攻击者只能访问所有加密数据，但对原始数据的域没有任何了解。

5. Design and Implementation

The protocol design for LunarX is meticulous about information protection(retrievability and confidentiality), public auditability and fault tolerance. As the incentive model that drives the operation of the network, blockchain technology is integrated for rewarding and punishment. Peers, providing computer resource to LunarX, are rewarded with tokens, while those who cheat, will be severed from the network, even their tokens will be deducted as punishment. In this section we take a more detailed look at each of these aspects.

5.设计和实施

LunarX的协议设计对信息保护（可检索性和机密性）、公众可审计性和容错性进行了精心设计。 作为推动网络运营的激励模式，区块链技术被集成进来用于奖励和惩罚。 向LunarX提供计算机资源的节点将获得奖励，而那些作弊者将被从网络中剔除，甚至他们的代币也将被扣除作为惩罚。 在本节中，我们将更详细地介绍这些方面。

5.1. Network

As mentioned before, LunarX network has three types participants: DApps and their users, trackers and service providers.

Trackers are a group that runs auditors to validate the authentication of data storage. Any individual who has computational resource can join the group, receive tasks of audition, issuing challenges to providers and verify their proofs. In auditing a huge amount of data, it is a resource consuming process, so officially we need the device connected to be an auditor meets some particular computing criteria.

5.1 网络

如前所述，LunarX网络有三种类型的参与者：DApp和其用户，追踪者，服务提供者。

追踪者是一个操作审核人员验证数据存储的组。 任何拥有计算资源的个人都可以加入该组，接受验证任务，向提供者发出请求并验证他们的证明。 在审核大量数据时，这是一个耗费资源的过程，因此我们需要将所连接的一些设备作为审核员来满足某些特定的计算标准。

Peers, who have spare resource, especially big disk and big I/O band width, contribute the capacity to LunarX in exchange for service rewards. They are organized in a DHT network, e.g Kademlia DHT [8] and its improvement S/Kademlia DHT [9], which stores nodes and resource locations throughout the network. Kademlia DHT uses XOR(exclusive or) operator to calculate the logical distance between any two peers. All nodes in a massive network according to the XOR metric construct a prefix binary tree, hence has the logarithmic complexity in searching through.

Suppose we have 30 million nodes, in the worst case, it only costs 25 hops to reach the searched node.

拥有多余资源，尤其是大磁盘和大I/O带宽的节点，为LunarX贡献了数据交易能力，获得服务奖励。 他们被组织在一个DHT网络中，例如Kademlia DHT [8]及其改进的S / Kademlia DHT [9]，存储整个网络中的节点和资源位置。 Kademlia DHT使用XOR（独占或）运算符来计算任何两个节点之间的逻辑距离。 根据XOR度量，一个海量网络中的所有节点构造一个前缀二叉树，因此在搜索时具有对数复杂度。

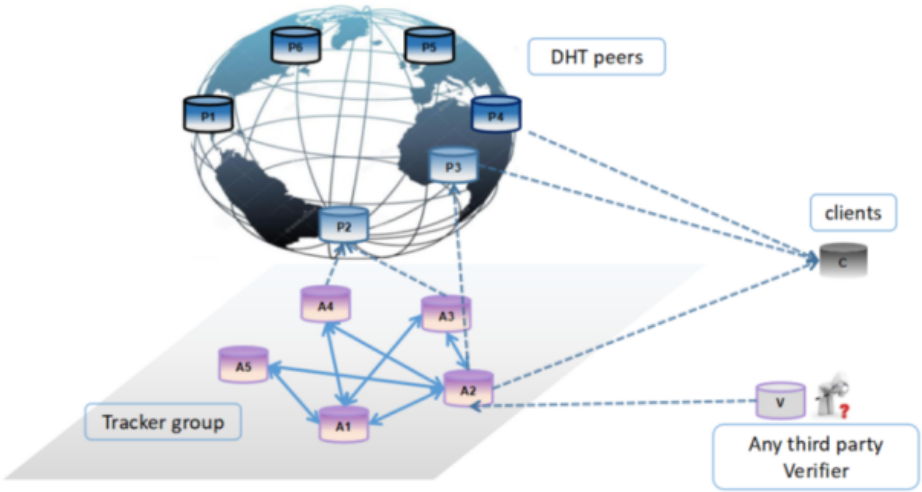

Figure 2. Network Topology

假设我们有3000万个节点，在最坏的情况下，它只用25跳就能到达搜索节点。

A much more encouraging advantage of Kademlia DHT is the ability against the DOS(denial of service) attack. Even if some of nodes is flooded, routing procedure will seek around the jammed area, therefor it will have limited effect on network availability.

DApps, who seek extra bigger and trustworthy data service from LunarX for their clients, have two basic requirements: record data and query(download if data is some kind of files) data. Before submitting data to LunarX, the owner has to encrypt it by a selected symmetric/asymmetric encryption method. If one loses his key (private key for asymmetric method or secret key for symmetric method), then he exposes his data to the public. The data D exists in LunarX is the encrypted version enc(D), which part we have already explained in the property of retrievability(P6, section 4.1).

Kademlia DHT的一个更令人鼓舞的优势是抵御DOS（拒绝服务）攻击的能力。 即使有些节点陷落，路由处理也会在受干扰的区域寻找，因此，攻击对网络可用性的影响是有限的。

DApps希望LunarX为其客户提供额外的更大和可靠的数据服务，它有两个基本要求：记录数据和查询（如果数据是某种文件，则下载）数据。 在向LunarX提交数据之前，数据拥有者必须选择对称/非对称加密方法对数据进行加密。 如果用户丢失密钥（私钥用于非对称加密，密钥用于对称加密），他的数据就公开暴露出来了。 LunarX中保存的数据D是加密版本enc（D）的数据，我们已经在可检索性属性（P6，第4.1节）中解释了这一部分。

Figure 2 illustrates the network topology, showing how trackers and providers are organized in LunarX. Tracker are labeled with Ai, i = 1, 2, 3....., service providers are those around the world, marked by Pj,j = 1,2,3..., as well as a node with label C is a user.

Accodring to publicly verifiability (P2, section 4.1), any third parties, labeled with V , will register themselves to the tracker group to take tasks of audition.

图2显示了网络拓扑结构，显示了LunarX中追踪者和提供者的组织方式。 追踪者用Ai标记，i = 1,2,3 ……，遍布全球的服务提供者，标记为Pj，j = 1,2,3 …，以及标签为C的节点是一个用户。 根据公开可验证性（P2，第4.1节），任何标有V的第三方都需要向追踪者组注册以执行审核任务。

Maps between data and nodes who store them, are maintained by tracker group in a big hash table. The trackers use data hashes as hash table keys. Each key maps to a unique node ID as the value which is responsible to store the corresponding data. Since storer nodes can dynamically join and leave the network, the hash table maintained by trackers is dynamic, periodically updated. When a node is not accessible for whatever reasons, the trackers who have the affected part of the hash table, have to update the table, replacing the dead nodes ID by another who has a copy of the data. Trackers themselves can be the backup nodes, if users choose them to be.

Trackers drive a bunch of verifiers to periodically audit peers for the validity of the data. Then also, every tracker maintains a list of registered verifiers.

数据和存储数据的节点之间的映射由追踪者组维护在一个大型哈希表中。 追踪者使用数据哈希作为哈希表键。 每个键映射到唯一的节点ID作为负责存储相应数据的值。 由于存储节点可以动态加入和离开网络，因此追踪者维护的哈希表是动态的，并且定期更新。 当某个节点因任何原因而无法访问时，哈希表受到影响的追踪者必须更新该表，将另一个有该数据副本的节点ID替换死亡节点ID。 如果用户选择，追踪者本身也可以成为备份节点。

追踪者驱动一批审核者定期审核节点数据的有效性。 然后，每个追踪者都维护一个已注册的审核者列表。

5.2. Client Protocol and Fault Recovery

DApps may have big data entries. For example, a P-2-P storage DApp has each data row which records users' files and metadata. As we mentioned before, this row storage sharding is straight forward. On client side, if one row has size less than 1Mb, we directly append it to the end of the table. If one entry is binary type and has bigger size, say 5GB, we will not store it in the table. We segment the entry data into each piece of 500MB, and record this segmentation and content links in the entry. These links will be used to patch up all the data pieces when the row is retrieved from LunarX. It is straight forward, then when we talk about entry data D, we suppose it is less than 500MB.

5.2 客户端协议和故障恢复

DApps可能拥有很大的数据量。 例如，P-2-P存储DApp用每一数据行记录用户的文件和元数据。 正如我们前面提到的那样，每一行直接存储数据分片。 在客户端，如果一行的大小小于1Mb，我们直接将它附加到表的末尾。 如果一个数据是二进制类型并且比较大，比如5GB，那么我们不会将它存储在表中。 我们将此数据分割成每个500MB的分片，我们记录分片数据和其内容链接。 当从LunarX中请求行数据时，这些链接将用于拼接所有数据分片。 这很简单，那么当我们谈论数据条目D时，我们假设它小于500MB。

To a globally distributed system, nodes are unstable. They may be disconnected, jammed, hacked or destroyed purposely. Some solutions save multiple replicas as data redundancy. One fails, gets another. While our design leverages Reed–Solomon erasure coding for recovering data from error nodes. It is able to detect and correct multiple symbol errors. Data D is divided into a list of m parts, and by adding t check parts to the list, a ReedCSolomon code can detect any combination of up to t erroneous parts, or correct up to ⌊t/2⌋ parts [13] [14].

对于全球化的分布式系统来讲，某些节点是不稳定的。 节点可能会断开连接、堵塞、黑客攻击或故意破坏。 有些解决方案是保存多个数据副本作为冗余数据。 一个没了，获取另一个。 LunarX的设计是利用Reed–Solomon抹除编码方案从错误节点恢复数据。 它能够检测和纠正多个符号错误。 数据D被分割成由m个分片组成的列表，通过将t部分的检查添加到列表中，ReedCSolomon代码可以检测到最多t个错误部分的任何组合，或纠错达到 ⌊t/2⌋ 个部分[13] [14]。

While data replication is still an option. User may choose to have a back up of his original data, but with different metadata, water prints for proof, timestamps and some other information that identify the data. Only user himself and the trackers know the two copies are actually the same. Replica can be stored on trusted node, for example, on the trackers. We will go back to this point in the next section.

数据复制仍然是一种选择。 用户可以选择备份他的原始数据，但具有不同的元数据，用于证明的水印，时间戳和用于识别数据的一些其他信息。 只有用户自己和追踪者知道这两个副本实际上是相同的。 副本可以存储在信任节点上，例如追踪者上。 我们将在下一节讲到这一点。

5.2.1. Insert into Table.:
Insert(Row={Col1 = D1 , Col2 = D2 , ...}):

1) Via LunarX client, user U encrypts Di to enc(Di), keeps his private key in safe.
2) Selects a code rate of error eraser coding.
3) For each column, LunarX client divides E(Di) = enc(Di) into pieces Ej = encj (Di), j = 1, 2, ..., n, constructs Reed–Soloman data blocks RSk (Ej) = encj (Di), k = 1, 2, ..., m, where m > n, depending on the code rate he chooses.
4) Injects in random positions pk (RSj), k ∈ Z in some of these pieces RSj with special water print ck(RSj).
5) LunarX tells U these ck(RSj) and pk(RSj).
6) Seek available storage peers via trackers for RSj.
7) LunarX client calls:
(key, P) = reserve(hash(RSj (E)), RSj (E)),
directly connects to these peers and transfer RSj to them by (key, P, address) = store(key, hash(RSj (E)), RSj (E)).

5.2.1 插入表格。：
插入（行= {Col1 = D1，Col2 = D2，...}）：
1）通过LunarX客户端，用户U将Di加密成enc（Di），把私钥保存在安全地方。
2）选择错误擦除编码的码率。
3）对于每一列，LunarX客户端将E（Di）= enc（Di）分成Ej = encj（Di），j = 1,2，...，n，构成Reed–Soloman数据块RSk（Ej）= encj（Di），k = 1,2，...，m，其中 m> n，取决于用

户选择的码率。
4）把特殊的水印ck（RSj）注入到这些片段RSj中的随机位置pk（RSj），k∈Z。
5）LunarX告诉用户U： ck（RSj）和pk（RSj）。
6）通过RSj的追踪者寻找可用的存储节点。
7）LunarX客户端调用：
（key，P）= reserve（hash（RSj（E））， RSj（E）），
通过（key，P，address）= store（key，hash（RSj（E））， RSj（E））直接连接到这些节点将RSj转发给它们。

Before Step 6), we have to mention that the client has already joined the DHT network with its NodeID, generated by hashing his public key, and download the DHT routing table. At the same time, it connects to known trackers for peers which are available, and have the space it needs.

In Step 7): (key, P) = reserve(hash(RSj (E)), RSj (E))

The reserve operator invokes trackers to reserve space for RSj(E), with its hash hash(RSj(E)) as the parameters, in response to a client request. The peer P generates a globally unique string which serves as a key that can be used by the client to insert and query his data RSj(E). Reserved space may expire if terms that both sides agree upon can not be satisfied. For example, after a certain time slot, P has not receive his payment, he may discard this reservation.

(key, P, address) = store(key, hash(RSj (E)), RSj (E))

The store operation sends RSj(E) and its hash directly to the peer P. The client must submit the key returned by reserve(hash(RSj (E)), RSj (E)) to authenticate with the peer. If some columns have big size data, the inserting procedure may be time consuming, LunarX supports broken-point continuingly transferring for big data segments.

在步骤6之前），我们已经提到客户端已经通过哈希公钥生成的NodeID加入了DHT网络，并下载了DHT路由表。同时，它连接到已知的可用节点追踪者，获得需要的空间。

在步骤7）中：（key，P）= reserve（hash（RSj（E））， RSj（E））

预留操作者调用追踪者来为RSj（E）预留空间，以其哈希 hash（RSj（E））作为参数，以响应客户端请求。节点P生成一个全局唯一的字符串，该字符串作为客户端可用于插入和查询其数据RSj（E）的密钥。如果双方同意的条款不能完整执行，保留的空间可能会过期。例如，在某个时间段之后，节点P还没有收到付款，他可能会作废此次保留。

（key，P，address）= store（key，hash（RSj（E））， RSj（E））

存储操作将RSj（E）及其哈希值直接发送到节点P。客户端必须提交由
reserve（hash（RSj（E））， RSj（E））函数返回的密钥，以便认证节点。如果某些列的数据量较大，插入过程可能会很耗时，LunarX支持大数据量的断点续传。

5.2.2. Append Index for Columns. Since privacy protection is the top priority of the system design, some of the indexes is commutative, others don't. Then as we known, inverted index and hash table can be stored remotely. In LunarX, users may choose to

store the index of their column data locally or delegate a provider to maintain their indexes. The delegated providers then become remote index service providers for their clients.

Since the storage is distributed, we have to use the separated architecture for a embedded data service engine in P–2–P network(figure 3).

5.2.2 为列添加索引。 由于隐私保护是系统设计的首要任务，因此一些索引是可交换的，其他索引则不是。 如我们所知，倒排索引和哈希表可以远程存储。 在LunarX中，用户可以选择在本地存储列数据的索引，或委托提供者维护其索引。 委托的提供者就成为客户的远程索引服务提供者。

由于存储是分布式的，我们必须在P–2–P网络中使用分离的架构来构建嵌入式数据服务引擎（图3）。

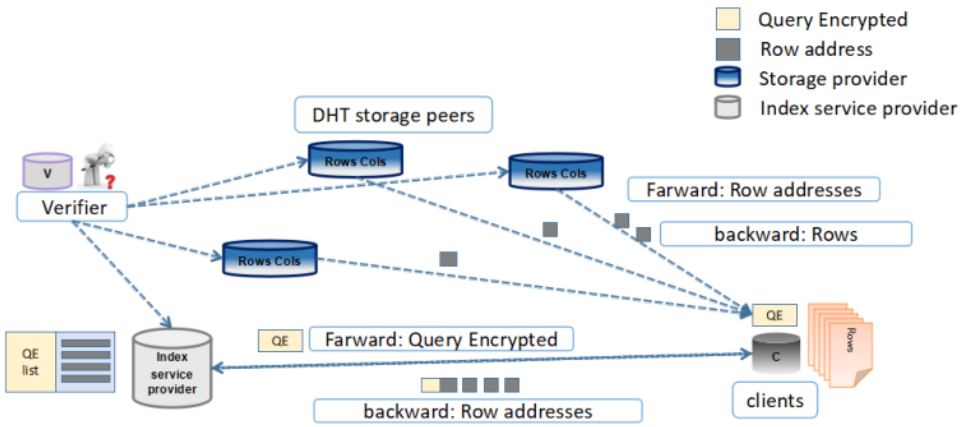

Figure 3. P-2-P Service Engine

Here we use fulltext index as an example. **AppendFulltextIndex**(Coli = Di):

1) For Data Di, Inser(Di) as in 5.2.1.
2) If Di is a document or any object with abstracted features, tokenize D or its features, then we get a list of words: list(w1, w2, ...).
3) Encrypt each words in the list, and get List(D) = list(E(w1), E(w2), ...).
4) Insert List(D) and the data address(key, P, address) returned by store(key,hash(RSj(E)),RSj(E)) into remote index service provider.
5) On the side of index provider, select a tokenizer that separates List(D) via simple comma, and index this content.
6) Return client the index provider ID, which will be used in search.

这里我们使用全文索引作为例子。 **AppendFulltextIndex**（Coli = Di）：

1）对于数据Di，Inser（Di）如5.2.1所示。
2）如果Di是一个文档或任何具有抽象特征的对象，标记D或其特征，则我们得到一个单词列表：list（w1，w2，...）。
3）对列表中的每个单词进行加密，得到List（D）= list（E（w1），E（w2），...）。

4）把 List（D）和 store（key，hash（RSj（E））， RSj（E））返回的数据地址（key，P，address）插入到远程索引服务提供者。
5）在索引提供者一侧，选择一个通过简单逗号分隔列表（D）的标记器，并索引这些内容。
6）返回用于搜索的索引提供者ID给客户端。

Normally, users are thin clients, having limited space for meta-data only. In contrast with centralized DB clusters which are heavy, LunarX partition the whole database into peers around the world, with each partition serving multiple clients(Figure 4). Since indexes are normally compressed structure, an index provider with mainstream desktop computer configuration may be able to serve tens of clients with each having 50 million records.

通常情况下，用户是瘦客户端，只有有限的空间存储元数据。 与中心化的数据库集群相比，LunarX将整个数据库分割为遍布全球的节点，每个节点为多个客户端提供服务（图4）。 由于索引通常是压缩的结构，一个有主流台式计算机配置的索引提供者，就可能服务于数千个客户，给每个客户提供多达5000万条记录的服务。

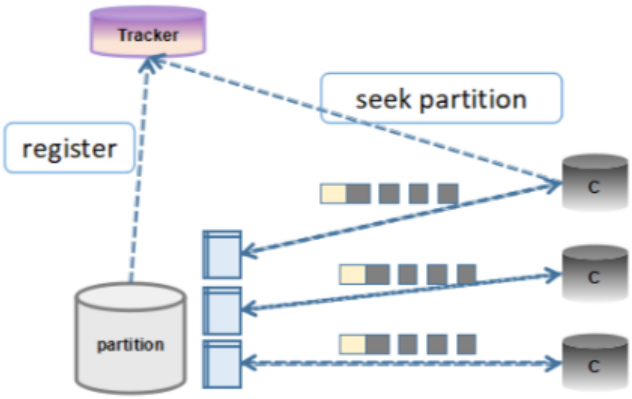

Figure 4. One to Many Partition

We must emphasize that only data with property of autonomous can be partitioned in the way LunarX does. For example, data from UGC(user generated content) DApps.

我们必须强调，只有具有自治属性的数据才能以LunarX的方式进行分区。 例如，来自UGC（用户生成内容）DApps的数据。

5.2.3. Query. :
Query(.):
1) connect to index provider, send query and get result set including the addresses of rows that match the particular query.
2) For each row D, which has been encoded in RSj (E), j = 1, 2, ..., m ， query DHT by calling retrieve(hash(RSj (E))), which locates peers that have the data.
3) Till enough RSj(E) have been found and downloaded, stops finding.
4) If a node holding any RSj(E) that is not accessible, just seek the next one RSj+1(E).
5) If the error node exceeds the upper bound of code rate, this file damages. Seek via trackers if there are replicas. If not, this row is lost. Procedure Ends.
6) Get E(D) by decoding RSj(E).
7) Decrypt E(D) to get D.

retrieve(hash(RSj (E))): Given the hash of RSj (E), the retrieve operation locates peers who have the data, and retrieves data from them.

5.2.3 查询

Query(.):

1）连接到索引提供者，发起查询并得到包含与特定查询匹配的行地址的结果集。
2）对于已经在RSj（E）中编码的每一行D，j = 1,2，...，m，通过调用定位节点数据的函数retrieve（hash（RSj（E））来查询DHT。
3）直到找到足够的RSj（E）并下载完成，才停止查找。
4）如果一个节点有任何不能访问的RSj（E），就开始寻找下一个RSj + 1（E）。
5）如果错误节点超过码率的上限，则说明该文件受损了。 通过追踪者查找是否有副本数据。如果没有，这一行就丢失掉了。 程序结束。
6）通过解码RSj（E）获得E（D）。
7）解密E（D）得到D.

retrieve（hash（RSj（E））：给定RSj（E）的哈希，检索操作定位具有数据的节点，并从中检索数据。

In the case of hostage attack briefed in section 3.3, clients may repeatedly challenge the peer, and each time get a piece of RSj(E). After sufficient times of challenging, clients will reconstruct the complete RSj(E). Hackers will intentionally use the same methodology to get the data, but what they get is the encrypted version of the original data, and the client(the owner) is the only one who holds the key for decryption.

在第3.3节所述的人质攻击案例中，客户端可能会反复请求节点，每次获得一段RSj（E）。 经过足够多次的请求后，客户端将能重建完整的RSj（E）。 黑客能使用相同的方法来获取数据，但他们得到的是原始数据的加密版本，而客户端（所有者）是唯一拥有解密密钥的人。

There is another way against the hostage attack. Just ignore RSj(E) and seek the next piece RSj+1(E), we will recover E as long as most of these pieces are correctly downloaded. This is what a download protocol does in above Query(.) procedure.

对付人质攻击还有另一种方法。 只要忽略RSj（E）并寻找下一个片段RSj + 1（E），只要大部分片段都被正确下载，我们就能恢复E。这是在上面一段Query（.）描述过程里的下载协议的功能。

5.3. Tracker Protocol

Trackers have lot of things to do, including: maintaining data hash table, synchronizing for global consistency, managing verifiers for audition, detecting health of nodes via heart-beating, etc. Among these functionality, we give the details of the most important two operations: resolvePeer(.) for finding the resource, and audit(.) for proving the data validity.

5.3 追踪者协议

追踪者有很多事情要做，包括：维护数据哈希表、同步全局一致性、管理审核者进行验证工作、通过心跳检测节点的健康状况等等。在这些功能中，我们给出最重要的两个操作的细节

： 用于查找资源的resolvePeer(.)，用于验证数据有效性的audit(.)。

(P, key) = resolvePeer(hash(RSj (E))):
This operation is invoked by a client to find a peer which is able to store the data piece RSj(E). The client sends the request to the tracker to which it is connected, with the hash of RSj(E) as a parameter. The tracker calculates the real position A according to this hash hash(RSj(E)) by some strategies, for example the consistent hashing [?] [?], then forward the request to it. When the request reaches the tracker A, it checks its list of registered storer nodes, and their capacity, QOS and credits to determine a qualified provider P for RSj(E).

(P, key) = resolvePeer(hash(RSj (E))):
该操作由客户端调用以找到能够存储数据片段RSj（E）的节点。 把RSj（E）的哈希值作为参数，客户端将请求发送到与其连接的追踪者。 追踪者通过一些策略（例如一致性哈希[?] [?]），根据哈希值hash（RSj（E））计算实际位置，然后将请求转发给客户端。 当客户端请求到达追踪者A时，追踪者检查其注册的存储器节点列表、节点容量，QOS和信用以确定RSj（E）的合格提供者P。

audit(Pi,hash(RSj(E)),ci) via PORI:
A tracker audits a peer Pi by issuing challenge ci, together with the hash hash(RSj(E)) of the data, to that peer. This procedure calls PORI that is defined in section 4.3 to proof the data possession. audit(.) will be invoked randomly and periodically as what we have explained before.

通过PORI 来调用 audit(Pi,hash(RSj(E)),ci)：
追踪者通过向该节点发出请求 ci 以及数据的哈希值 hash（RSj（E））来审核节点Pi。 此过程调用4.3节中定义的PORI来证明数据所有权。 audit(.) 将按照我们之前解释的那样随机且周期性地调用。

6. Applications

Based on LunarX, we have several DApps in developing. The first is a P-2-P storage application, users store their documents via a light client. And the second is in healthcare field, via smart devices, users submit their daily training data to LunarX, get analysis report of his health condition, share the parts they feel proud of and keep other data in secrete. Every one has a great amount of private data generated during his daily life. Due to the locality of LunarX, the overhead is well balanced around the world.

6.应用程序

基于LunarX，我们有几个DApps正在开发中。 第一个是P-2-P存储应用，用户通过轻客户端存储他们的文档。 第二是在医疗领域，通过智能设备，用户将他们的日常训练数据提交给LunarX，获得健康状况的分析报告，分享想要分享的部分，其余的会安全的保存。 每个人在日常生活中都会产生大量的私人数据。 基于LunarX的数据分布的局部性，系统开销会平衡到全球的节点上。

7. Future Work

In current release, LunarX supports mainstream local file systems, like NTFS for windows, ext family on linux, or mac file systems, both of desktop and server. We may offer versions for professional users that file systems for cluster, e.g. HDFS [32], will also be supported.

7.未来的工作

在当前版本中，LunarX支持主流本地文件系统，如Windows的NTFS、Linux上的ext系列、台式机以及服务器的mac文件系统。 我们可以为专业用户提供集群文件系统的版本，例如HDFS [32]也将得到支持。

In the future, we will have more DApps for various businesses, which will greatly benefit from the scalability of LunarX. On fundamental research, we will invest great resource on distributed and encryption commutative data structures, with a much more practical view of how to balance the efficiency, scalability, privacy, how to design more available data functionality that DApps can smoothly plugin.

未来，我们将为各种业务提供更多DApp，这将从LunarX的可扩展性中受益匪浅。 在基础研究方面，我们将投入大量资源用于分布式和加密可交换数据结构，并提供更实用的观点，以便平衡效率、可扩展性、隐私性，以及设计更多可用的数据功能让DApps可以平滑的即插即用。

Acknowledgments

LunarX is inspired by many brilliant existing works, not only the works we mentioned in the reference list, but also our technical team and many of the online-offline discussions that we can hardly list all of them here. As an engineering pilot in this field, we hope this work to be an entry point for building a community that experts, scholars, geeks and whoever have enthusiasm can work together to advance the technology.

致谢

LunarX的灵感来自于许多很棒的现有工作成果，不仅限于我们在参考列表中提到的，还有我们的技术团队以及许多在线和离线讨论的，我们无法在此列出所有作品。 作为这一领域的实验性项目，我们希望这项工作成为建立一个由专家、学者、极客和任何有激情的参与者组成的社区的切入点，可以共同推动这项技术的发展。

References

参考文献

[1] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. http://bitcoin.org/bitcoin.pdf

[2] Blockchian: https://en.wikipedia.org/wiki/Blockchain

[3] Publicly Verifiable Secret Sharing, http- s://en.wikipedia.org/wiki/Publicly Verifiable Secret Sharing

[4] Juan Benet, IPFS-Content Addressed, Versioned, P2P File System, https://ipfs.io/ipfs/

[5] Filecoin: https://filecoin.io/filecoin.pdf

[6] Sia: https://sia.tech/sia.pdf

[7] Storj: https://storj.io/storj.pdf

[8] Kademlia (DHT): https://en.wikipedia.org/wiki/Kademlia

[9] I.Baumgart, S.Mies, S/Kademlia: A practicable approach towards se- cure key-based routing, in: Parallel and Distributed Systems, 2007 International Conference on, volumn 2, pages 1-8, IEEE, 2007.
[10] Merkle Trees: https://link.springer.com/content/pdf/10.1007/978-3- 540-24676-3 32.pdf
[11] Karger.D,Lehman.E,Leighton.T ,Panigrahy.R,Levine.M,Lewin.D,(1997). Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing. ACM Press New York, NY, USA. pp. 654C663.
[12] Consisting Hashing. https://en.wikipedia.org/wiki/Consistent hashing
[13] ReedCSolomon error correction, https://en.wikipedia.org/wiki/Reed- Solomon error correction
[14] Reed, Irving S.; Solomon, Gustave (1960), Polynomial Codes over Certain Finite Fields, Journal of the Society for Industrial and Applied Mathematics (SIAM),8(2): 300C304.
[15] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. Fully Homomorphic Encryption without Bootstrapping. In ITCS 2012.
[16] Z. Brakerski and V. Vaikuntanathan. Efficient Fully Homomorphic Encryption from (Standard) LWE. In FOCS 2011 (IEEE)
[17] Z. Brakerski. Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. In CRYPTO 2012 (Springer)
[18] A. Lopez-Alt, E. Tromer, and V. Vaikuntanathan. On-the-Fly Mul- tiparty Computation on the Cloud via Multikey Fully Homomorphic Encryption. In STOC 2012 (ACM)
[19] C. Gentry, A. Sahai, and B. Waters. Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. In CRYPTO 2013 (Springer)
[20] R.Agrawal, J.Kiernan, R.Srikant, Y.Xu. Order preserving encryption for numeric data, in Proceeding SIGMOD '04 Proceedings of the 2004 ACM SIGMOD international conference on Management of data, Pages 563-574
[21] D.R.Stinson. Cryptography: Theory and Practice. CRC Press, 2nd edtion, 2002.
[22] Knuth, D. E. (1997) [1973]. "6.5. Retrieval on Secondary Keys". The Art of Computer Programming (Third ed.). Reading, Massachusetts: Addison-Wesley. ISBN 0-201-89685-0.
[23] Proof of data possession, http://cryptowiki.net/index.php?title=Proof of data possession
[24] B.Cohen. Incentives build robustness in bittorrant. in Proceedings of the 1st Internation Workshop on Economics of P2P Systems (P2PECON '03)
[25] Protocol Labs. Technical Report: Proof of Replication. 2007
[26] Bayer, R.; McCreight, E. (1972), "Organization and Maintenance of Large Ordered Indexes" (PDF), Acta Informatica, 1 (3): 173C189, doi:10.1007/bf00288683
[27] Merkle DAG: https://github.com/ipfs/specs/tree/master/merkledag
[28] Tommi Virtanen. Git for Computer Scientists: http://eagain.net/articles/git-for-computer-scientists/
[29] A.Juels, B.S. Kaliski Jr, PORs: Proofs of Retrievability for Large Files, in Proceeding CCS '07 Proceedings of the 14th ACM conference on Computer and communications security, Pages 584-597 .
[30] H.Shacham, B.Waters, Compact Proofs of Retrievability, Journal of Cryptology, July 2013, Volume 26, Issue 3, pp 442C483.

[31] G.Ateniese,R.Burns, etc, Provable Data Possession at Untrusted S– tores, CCS07, October 29CNovember 2, 2007, Alexandria, Virginia, USA, Page 598–610
[32] Hadoop Distributed File System, https://hadoop.apache.org/docs/r1.2.1/hdfs design.html
[33] Sybil attack, https://en.wikipedia.org/wiki/Sybil_attack
[34] Douceur, John R., The Sybil Attack, International workshop on Peer– To–Peer Systems. Retrieved 23 April 2016.
[35] V.Buterin,etc, Etheruem White Paper, https://github.com/ethereum/wiki/wiki/White –Paper