TPRE-分布式门限代理重加密

文章学习：TPRE-分布式门限代理重加密

数学部分

一般情况下，为节约本地资源，我们会把文件存储至云服务提供商。但是由于大部分的云服务供应商并不能完全值得信任，可能会在未经用户允许的的情况下，擅自泄露用户上传的隐私数据或重要文件。

解决这个问题最直接的方法是数据加密。对于涉及用户隐私或含有敏感信息的数据文件，将数据加密后上传是一种普遍的保证数据机密性的方法，只需数据上传者保管好解密密钥。

但云计算服务中存在大量的需要共享数据的应用需求，而这样的加密方式并不能将数据通过云服务器共享给他人，因为只有数据上传者能够解密文件。因此，我们需要一种新的协议，将上传到云服务器中的密文进行安全有效的转换，使得被授权者也能解开密文。

混合加密机制

由于公钥密码是运行在代数计算上的密码算法，其计算效率远低于对称密码，因此，在待加密数据量较大时，使用公钥密码直接对数据加密不是一个良好选择。在该场景中，可使用混合加密体制KEM/DEM：

- KEM是指密钥封装机制(Key-Encapsulation Mechanism)
- DEM是指数据封装机制(Data-Encapsulation Mechanism)，可以看做是对称加密方案。

这两个机制联合使用，可以提供数据的加解密效率，在密文需要传输时，也可降低通信开销。具体而言，DEM机制是用作保护原始数据，即使用对称加密算法，对原始数据进行加密保护；KEM是用作保护加密原始数据时所使用的对称密钥，使用公钥加密算法对对称密钥进行加密保护。

1. 代理重加密(Proxy Re-Encryption)

PRE是一种公钥密码方案，广泛应用于访问控制、分布式管理系统、加密邮件转发系统、垃圾邮件过滤系统等。它允许代理将一个公钥加密成的密文转换到另一个公钥加密成的密文，而代理则不能了解关于原始消息的任何信息。要做到这一点，代理必须拥有一个重加密密钥。

一个代理重加密算法通常包含三种角色，即数据拥有者Alice、数据接收者Bob和代理计算Proxy。算法的具体步骤如下：

假设数据m已经被Alice使用自己的公钥pk_A加密为密文$c := E(pk_A, m)$，并存储在半诚实Proxy中。

- Alice是数据拥有者，想要把数据授权给Bob，则需为Proxy生成重加密密钥$rk := RG(sk_A, pk_B)$。(r是前缀re)
- Proxy接收到rk后，对密文c进行重加密，得到新的密文$c' := RE(rk, c)$，并发送给Bob。
- Bob使用自己的私钥sk_B对c'进行解密，即可得到明文数据$m := D(sk_B, c')$。

这里想起来了一个基于NTRU的代理重加密方案：**NTRUReEncrypt: An Efficient Proxy Re-Encryption Scheme Based on NTRU**，下面具体介绍一下方案：

> 代理重加密方案：NTRUReEncrypt

委托人：A，被委托人：B，代理：proxy

- 密钥生成

下面再介绍一种UMBRAL代理重加密算法：

代理重加密适合在云计算场景中使用，即代理节点为计算性能较强的单节点，这与现有隐私计算体系架构不符，因为现在隐私计算架构通常是分布式架构。因此，需要对传统代理重加密方案进行改造，使之能够适应分布式计算环境。

分布式代理重加密是指将传统代理重加密中的单Proxy节点，拆分为多个Proxy节点。因而在对数据重加密时，需要多个Proxy节点参与合作计算。

考虑到选取参与计算的Proxy节点的灵活性，需要将分布式代理重加密重新设计为基于门限的分布式代理重加密。

2. Shamir秘密共享方案

Shamir Secret Sharing是一种加密技术，可以将秘密信息分散成多个部分，并分配给不同的人，只有所有部分被汇集在一起才能重构出原来的秘密信息。它是由以色列密码学家Adi Shamir在1979年发明的，被广泛应用于保护机密信息，例如在密码学、数字签名、多方计算等领域。其基本思想是通过多项式插值来实现信息的分散和重构，具有高度的安全性和可靠性。

Shamir密秘分享：假设有一个秘密信息，例如一个密码或者一个私钥，需要将这个秘密信息分配给两个人，可以使用Shamir Secret Sharing算法来实现。

3. 分布式门限代理重加密

TPRELib算法

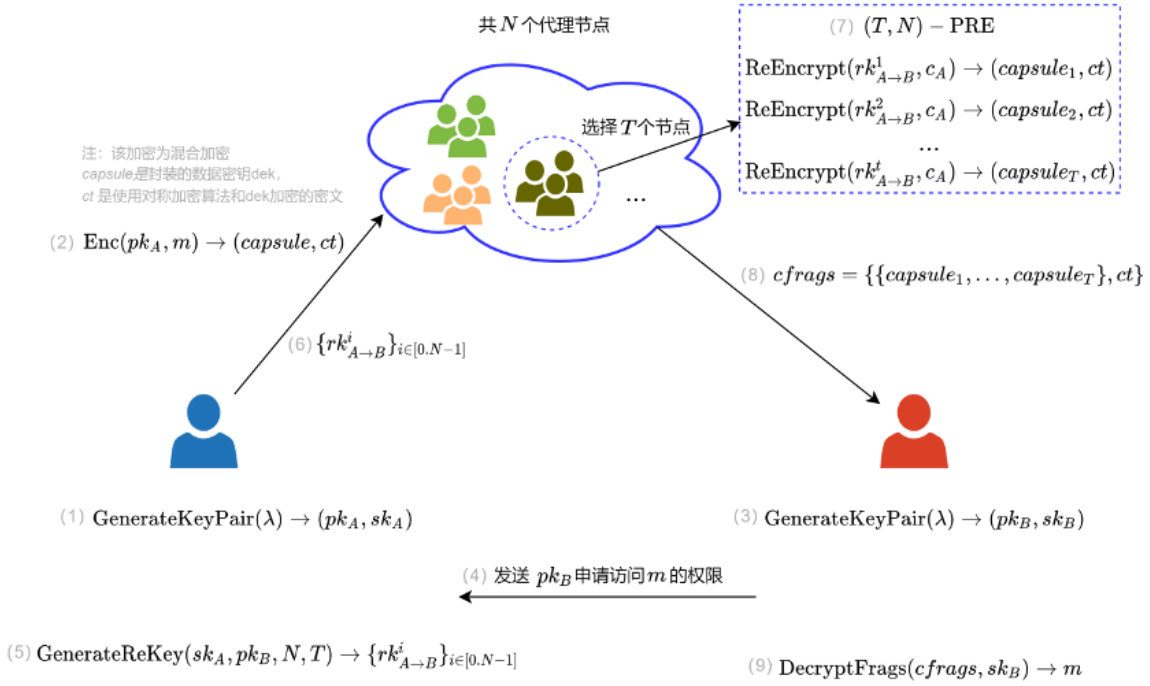
TPRELib共包含6个算法，分别为密钥对生成算法、重加密密钥生成、加密、解密算法、重加密算法、解密重加密密文算法。

假设数据持有方为Alice，接收方为Bob，代理者（节点）为proxy。

- Generate_TPRE_Key_Pair(λ)：输入安全参数λ，Alice和Bob各自生成公私钥对(pk_A, sk_A)、(pk_B, sk_B)；
- Generate_Re_Key$(sk_A, pk_B, N, t) \rightarrow \{rk_i\}, i \in [1, N]$：输入Alice的私钥，Bob的公钥，所有的代理节点数N以及门限t，输出重加密密钥集合。这里i是代理节点的ID；
- Encrypt$(pk_A, m) \rightarrow c$：输入Alice的公钥和明文m，输出密文c；

> 这里并不是直接使用pk_A加密明文，会导致性能过低的问题。在底层加密时，使用的是**对称加密算法**，在本库中对称加密算法由SM4实现，其对称密钥是在加密过程中随机产生，对称加密密钥由公钥加密保护【即：明文使用SM4加密，pk_A加密对称加密的密钥】
>
> 生成对称密钥时，需要用到密码哈希函数构建的**密钥派生函数KDF（Key derivation function）**，本库使用SM3代替SHA-2等国际算法，实现该KDF函数。【可以实现指定长度的哈希字符串】

- Decrypt$(sk_A, c) \rightarrow m$：Encrypt的逆过程；
- Re_Encrypt$(rk_i, c) \rightarrow c_i'$：代理节点输入重加密密钥和密文，输出新的密文
- Decrypt_Frage$(c_i'(i \in [t, N], sk_B)) \rightarrow m$：



算法部分

假设Alice为数据所有方，Bob为数据需求方，代理者（代理节点）为proxy

1. 密钥对生成

Alice在Z_q中均匀地随机选择a，计算g^a并输出Alice的密钥对$(pk_A,sk_A)=(g^a,a)$，Bob同样输出密钥对$(pk_B,sk_B)=(g^b,b)$。

2. ***重加密密钥生成 Generate_Re_Key$(sk_A,pk_B,N,t)\to\{rk_i\},i\in[1,N]$

- 随机抽样$x_A\in Z_q$并计算$X_A=g^{x_A}$
-

编程部分