

抗量子加密软件设计 - MoonKey

目录

1 作品概述	3
1.1 背景分析	3
1.1.1 后量子密码学	3
1.1.2 基于格的公钥加密算法NTRU	4
1.2 本作品的工作	5
1.3 特色描述	6
1.4 前景分析	7
2 设计与实现方案	7
2.1 实现原理	7
2.1.1 SM4算法	7
2.1.1.1 加密算法	7
2.1.1.2 密钥生成算法	9
2.1.1.3 解密算法	10
2.1.2 SM3算法	10
2.1.2.1 符号定义	10
2.1.2.2 常量与函数定义	11
2.1.2.3 算法描述	11
2.1.3 NTRU算法	13
2.1.3.1 相关参数定义与取值	14
2.1.3.2 密钥生成阶段	14
2.1.3.3 加密阶段	14
2.1.3.4 解密阶段	15
2.2 方案实现	15
2.2.1 大分组分组密码LBC实现	15
2.2.1.1 加密算法	15
2.2.1.2 密钥分配	17
2.2.1.3 解密算法	17
2.2.2 数字信封	18
2.2.3 明文分组	19
2.2.4 密文头部处理	21
2.2.5 软件目录结构	21

3 系统测试与结果	22
3.1 测试方案	22
3.2 算法测试	23
3.2.1 NTRU测试	23
3.2.2 SM4测试	24
3.2.3 LBC测试	24
3.2.4 CBC工作模式测试	25
3.2.5 ECB工作模式测试	26
3.2.6 API测试	26
3.3 功能测试	27
3.4 性能测试	34
3.4.1 测试方案	34
3.4.2 测试数据与结果	34
4 作品总结	36

1 作品概述

1.1 背景分析

1.1.1 后量子密码学

Shor的量子因式分解算法和其他一些高效的量子算法打破了许多经典的密码系统。作为回应，人们提出了基于计算问题的后量子密码学，即使对于量子计算机来说，这些问题也被认为很难解决[2]。

后量子密码学（Post-Quantum Cryptography, PQC）是密码学的一个领域，专注于开发能够抵抗量子计算机攻击的加密算法。随着量子计算机技术的进步，传统的公钥密码系统（如RSA和ECC等）面临着被量子计算机轻易破解的风险。后量子密码学旨在确保即使量子计算机变得强大，这些新设计的算法仍能保持数据的安全性。

后量子密码学的核心挑战是在不牺牲机密性的情况下满足对密码可用性和灵活性的需求[1]。以下是后量子密码学算法的几类主要研究方向：

1. 基于格的密码学（**Lattice-based Cryptography**）：量子计算的出现有可能打破许多经典的密码方案，导致公钥密码学的创新，这些创新侧重于后量子密码学原语和抗量子计算威胁的协议。基于格的密码学是一个有前景的后量子密码学家族，无论是在基础属性方面，还是在应用于加密、数字签名、密钥交换和同态加密等传统和新兴安全问题方面[3]。

它的应用在传统安全问题（如密钥交换和数字签名）和新兴安全问题（例如同态方案、基于身份的加密，甚至对称加密）方面都在激增[4]。基于格子的加密原语和协议提供了一组丰富的原语，可用于应对跨不同计算平台（例如云与物联网生态系统）部署以及不同用例带来的挑战，包括通过为基于非对称密钥加密的协议提供强大的基来对加密数据进行计算，以抵御强大的攻击者（使用量子计算机和算法），并提供超越传统加密范围的保护[1]。

代表算法：Kyber、NTRU、FrodoKEM。

2. 基于编码的密码学（**Code-based Cryptography**）：基于编码的密码学是为数不多的能够构建公钥密码系统的数学技术之一，这些公钥密码系统能够抵御配备量子计算机的对手。McEliece公钥加密方案及其变体是后量子公钥加密标准的候选者[6]。

这些算法利用编码理论中的难题，最著名的是基于纠错码的随机线性码译码问题。自1978年由Robert McEliece提出以来，McEliece加密系统仍未被量子计算机攻击成功。其利用普通Goppa码存在快速解码算法而普通线性码不存在快速解码

的事实，构建了一个公钥密码系统。该密码系统安全性高，同时允许极快的数据速率[7]。

代表算法：Classic McEliece。

3. 基于多变量多项式的密码学（**Multivariate Polynomial Cryptography**）：这些算法基于求解多元多项式方程组的难题。量子计算机在处理这些问题时没有显著的优势。多变量密码学适用于构建高效的数字签名方案[11]。

代表算法：Rainbow。

4. 基于哈希的密码学（**Hash-based Cryptography**）：这种方法使用哈希函数构建安全的加密和签名方案，尤其适合数字签名。哈希函数的安全性对量子计算机影响较小，因为量子计算机只能在特定程度上加速破解哈希（通过Grover算法加速碰撞攻击），但不会完全破坏哈希函数的安全性。相对简单，理论上安全性较高。

代表算法：SPHINCS+。

5. 基于同源映射的密码学（**Isogeny-based Cryptography**）：这类算法依赖于椭圆曲线同源映射的难题，属于较新的研究方向。它们具有非常小的密钥尺寸，是当前研究的热点之一。

代表算法：SIKE（Supersingular Isogeny Key Encapsulation）。

美国国家标准与技术研究院（NIST）于2016年启动了一项全球性的后量子密码标准化竞赛，目的是挑选出能够在量子计算机时代广泛应用的密码算法。经过几轮的候选算法评估，NIST在2022年宣布了最终阶段的候选算法，并计划在2024年左右发布首批后量子密码标准。

密钥封装算法（KEMs）：Kyber被选为NIST的主要密钥封装算法。

数字签名算法：Dilithium、Falcon和SPHINCS+被选为数字签名方案。

而其中Kyber、Dilithium和Falcon算法都是基于格的密码学问题，由此可见格密码在抗量子算法中有着显著优势。本作品在密钥生成阶段应用了基于格的抗量子算法NTRU。

1.1.2 基于格的公钥加密算法NTRU

NTRU（Number Theoretic Research Unit）是基于格的密码学（Lattice-based Cryptography）中的一种公钥加密算法。它由 Jeffrey Hoffstein、Jill Pipher 和 Joseph H. Silverman 于 1996 年提出，是最早提出的基于格的加密方案之一[8]。NTRU 的设计目标是提供既安全又高效的加密方案，能够抵抗量子计算机的攻击。

NTRU 的安全性依赖于格中困难问题，尤其是短向量问题（SVP）和最短向量解（SIS）。格问题在经典计算机和量子计算机上都被认为是非常困难的，尤其是当格维度很高时，这使得 NTRU 被认为具有抗量子攻击的能力[5]。

NTRU 是基于多项式环上的算术运算进行构建的。简单来说，它通过模一个大整数和一个多项式环来进行加密和解密操作。

1. **密钥生成。** NTRU 首先需要生成一个公钥和一个私钥。其关键步骤是生成两个多项式，分别是私钥 f 和一个辅助多项式 g ，然后通过某种方式将这些多项式组合起来生成公钥 h 。
2. **加密。** 加密时，发送方使用接收方的公钥 h ，将明文多项式进行混淆加密。加密的过程中，发送方还生成一个随机的掩码多项式来进一步增强安全性。
3. **解密。** 解密时，接收方使用其私钥 f ，将密文多项式还原为明文。NTRU 的设计确保只有拥有私钥 f 的用户能够正确解密得到原始明文。

NTRU 有一些显著的优点，使其在后量子密码学领域中脱颖而出：

1. **抗量子安全性** NTRU 基于格问题，而这些问题目前在经典计算机和量子计算机上都难以有效解决。因此，NTRU 被认为具有抵抗量子计算机的能力，这使得它成为后量子密码学的一个重要候选方案。
2. **高效的加密解密性能** NTRU 的加密和解密运算基于简单的多项式乘法和模运算，与 RSA 和 ECC 等传统公钥加密算法相比，NTRU 的计算速度更快。特别是在硬件中实现时，NTRU 的效率优势更加显著。
3. **相对较小的密钥尺寸** 与其他抗量子加密方案（如 McEliece 加密系统）的巨大密钥尺寸相比，NTRU 的密钥尺寸相对较小，这使得它在需要较小存储和快速传输的环境中表现优异。
4. **广泛应用的潜力** NTRU 由于其效率和安全性，特别适合在嵌入式设备、物联网（IoT）设备、移动设备等资源有限的环境中使用，包括 Palm 计算平台、高级 RISC 机器 ARM7TDMI、运动寻呼机研究，最后是 Xilinx Virtex 1000 系列 FPGA。在这些平台上，NTRU 都提供了卓越的性能，实现了一系列新的应用[10]。

1.2 本作品的工作

本作品着力于实现一款大分组分组密码算法，并将其用于文件加密，其主要流程为：

1. **背景假设:** 用户需要在Windows PC机上对特定文件加密，软件接受用户输入的文件后，返回密文文件和密钥文件，供用户在任意具备软件环境的Windows PC机上实现文件解密。
2. **文件加密:** 软件使用BCrypt库生成符合密码标准的随机数，进而随机生成符合用户要求的密钥。软件对文件使用PKCS#7进行填充，基于Feistel结构，通过对SM3和SM4的组合运用，对用户传入的文件进行分组加解密。
3. **密钥封装:** 考虑到直接将密钥原件写入文件的安全隐患，软件使用基于格的NTRU抗量子算法，对密钥进行封装。公私钥对为用户自行商定以此为保证软件的输出文件。
4. **软件实现:** 软件界面使用python简单GUI库，并用Cython库实现python和C++的接口，为保证有较高加解密效率，软件的内核以及密文密钥文件写入由C++进行实现，最终即成为一个可执行文件。
5. **软件封装:** 通过使用了Inno Setup软件将可执行文件进一步封装成软件安装包，其中加入了对下载用户的注册表修改，为我们生成的新后缀文件(.lkey与.lenc)进行了图标的创建。

1.3 特色描述

本作品实现的方案具有以下特点：

1. **安全性强且效率高:** 软件内核使用了组合式大分组的加密方式，以512bit为一个分组单元，组合使用SM3和SM4加密算法，在加密算法中实现抗量子的目标，保证了文件的安全性。
2. **软件密钥支持最高512bit,** 该密钥不会被软件直接写入文件，而是先用基于格的NTRU抗量子算法，对密钥进行封装，而算法中的公私钥对，则由用户自己商定，不直接暴露密钥，保证了密钥文件的安全性。
3. **安全性保障:** 与此同时，软件加密完全由本地文件提供，不涉及服务器或者恶意第三方，因而不会有恶意公私钥对或对称密钥的收集，不会出现泄露隐私的情况。
4. **NTRU算法的效率:** NTRU算法相比较其他格密码如Lyubashevsky、Peikert和Regev LPR-LWE的基于环LWE的方案，在保证基础安全性的前提下，效率较高。格密码算法在保护数据安全方面具有显著的优势。因为其不仅能够抵御量子计算机的攻击，还具有高效性和可扩展性[9]。

5. 适用范围广泛且软件操作简单:

1.4 前景分析

传统的公钥加密方案，如RSA、ECC和DSA等，依赖于特定数学问题的难解性。而量子科技的飞速发展使得大规模量子计算机的实现只是时间问题，一些量子算法的提出，如Shor量子算法[15]，使得对现代密码体制（公钥密码和对称密码）实施量子计算攻击成为可能，从而严重威胁经典密码的安全[19]。这一威胁严重影响了部分领域的行业，甚至是国家政府等领域。为了解决这一威胁，人们选择创建后量子密码技术。

抗量子密码具有广阔的应用前景，混合加密是当前后量子密码应用的主要形式。网络中，传输层安全协议（TLS）和安全外壳协议（SSH）优先采用后量子密码保护。与此同时，后量子密码与量子密钥分发融合应用趋势显著[20]。同时，政府和军事机构也将依赖抗量子技术来保护敏感信息免受未来量子计算机的攻击。在物联网和云计算场景下，抗量子加密算法的高效实现，尤其是全同态加密和基于哈希的签名方案，能够为海量数据提供加密保护。

我们本次以抗量子密码算法为背景设计了对文件进行加解密的密码算法，高效的加解密以及不同的分组模式可以应对大量的实际需求，对密钥的二次加密使用了NTRU算法，仍然保持抗量子的能力，提供了安全可靠的加解密软件，为那些具有安全性和高效性要求的场景提供了可靠的软件支撑。更多作品产出能够以点带面，加快后量子密码技术研发平台搭建。强化政府、社会组织、企业、科研机构在后量子密码技术建构上的联动协同，加强学术交流与合作，共同推进科研成果转化[18]。

2 设计与实现方案

2.1 实现原理

2.1.1 SM4算法

SM4分组密码算法（原名SMS4）于2006年公开发布。随着我国密码算法标准化工作的开展，SM4算法于2012年3月成为国家密码行业标准[13]，2016年8月发布成为国家标准4，2021年6月纳入ISO/IEC国际标准。

SM4算法采用非平衡Feistel结构，分组长度为128-bit，密钥长度为128-bit，迭代轮数为32轮。下面依次介绍SM4算法的加密算法、轮密钥生成方案和解密算法。

2.1.1.1 加密算法

SM4的加密算法如下所示。

1. 输入的128-bit明文 m , 按32-bit划分, 记为 $(X_0^0, X_0^1, X_0^2, X_0^3)$ 。
2. 进行32轮迭代, 每一轮有一个32-bit的轮密钥 K_{i-1} 参与运算。第 i ($i = 1, \dots, 32$) 轮的运算如下:

$$X_i^0 = X_{i-1}^1,$$

$$X_i^1 = X_{i-1}^2,$$

$$X_i^2 = X_{i-1}^3,$$

$$X_i^3 = X_{i-1}^0 \oplus T(X_{i-1}^1 \oplus X_{i-1}^2 \oplus X_{i-1}^3 \oplus K_{i-1})$$

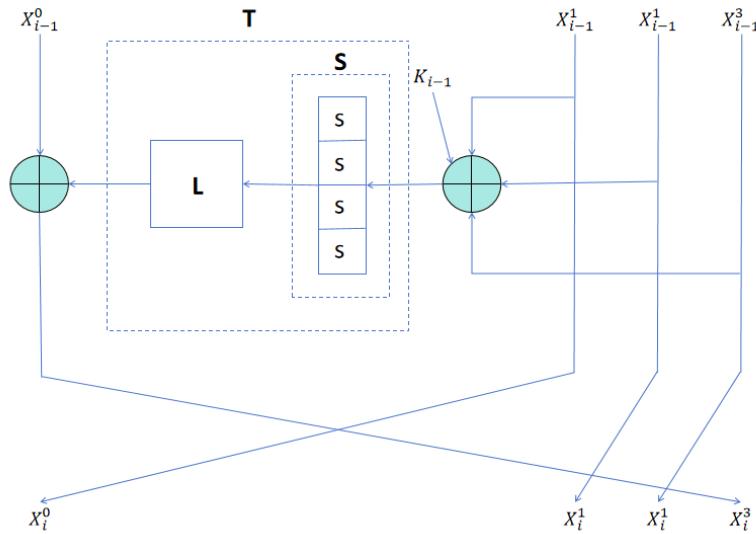


图 1: SM4的轮函数

3. 对最后一轮迭代的输出结果进行反序变换 R , 得到128-bit密文 c , 即

$$c = (Y^0, Y^1, Y^2, Y^3) = R(X_{32}^0, X_{32}^1, X_{32}^2, X_{32}^3) = (X_{32}^3, X_{32}^2, X_{32}^1, X_{32}^0).$$

其中, 函数 $T : \mathbb{F}_2^{32} \rightarrow \mathbb{F}_2^{32}$, 由非线性变换 S 和线性变换 L 复合而成, 即

$$T = L \circ S.$$

设函数 T 的32-bit输入为 $A = (a_0, a_1, a_2, a_3) \in (\mathbb{F}_{2^8})^4$, 则

- 非线性变换 S : 对每个字节单独进行查表代替操作, 即

$$B = (s(a_0), s(a_1), s(a_2), s(a_3)).$$

与AES类似, SM4算法采用1个S盒, 定义如1所示。在查表时, 将每个字节的高4-

bit看作行标，低4-bit看作列标，对应取值即为输出字节。例如，

$$s(0x07) = 0xB7.$$

表 1: SM4的S盒

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	D6	90	E9	FE	CC	E1	3D	B7	16	B6	14	C2	28	FB	2C	05
1	2B	67	9A	76	2A	BE	04	C3	AA	44	13	26	49	86	06	99
2	9C	42	50	F4	91	EF	98	7A	33	54	0B	43	ED	CF	AC	62
3	E4	B3	1C	A9	C9	08	E8	95	80	DF	94	FA	75	8F	3F	A6
4	47	07	A7	FC	F3	73	17	BA	83	59	3C	19	E6	85	4F	A8
5	68	6B	81	B2	71	64	DA	8B	F8	EB	0F	4B	70	56	9D	35
6	1E	24	0E	5E	63	58	D1	A2	25	22	7C	3B	01	21	78	87
7	D4	00	46	57	9F	D3	27	52	4C	36	02	E7	A0	C4	C8	9E
8	EA	BF	8A	D2	40	C7	38	B5	A3	F7	F2	CE	F9	61	15	A1
9	E0	AE	5D	A4	9B	34	1A	55	AD	93	32	30	F5	8C	B1	E3
A	1D	F6	E2	2E	82	66	CA	60	C0	29	23	AB	0D	53	4E	6F
B	D5	DB	37	45	DE	FD	8E	2F	03	FF	6A	72	6D	6C	5B	51
C	8D	1B	AF	92	BB	DD	BC	7F	11	D9	5C	41	1F	10	5A	D8
D	0A	C1	31	88	A5	CD	7B	BD	2D	74	D0	12	B8	E5	B4	B0
E	89	69	97	4A	0C	96	77	7E	65	B9	F1	09	C5	6E	C6	84
F	18	F0	7D	EC	3A	DC	4D	20	79	EE	5F	3E	D7	CB	39	48

- 线性变换 L:对非线性变换 S的 32-bit输出 B，计算

$$L(B) = B \oplus (B \lll 2) \oplus (B \lll 10) \oplus (B \lll 18) \oplus (B \lll 24).$$

其中， \lll 表示循环左移 x -bit。

2.1.1.2 密钥生成算法

SM4算法的主密钥 k 为 128-bit，轮密钥生成方案也采用了非平衡Feistel结构，具体操作如下：

1. 将 k 按 32-bit 进行划分，记为 (MK_0, MK_1, MK_2, MK_3) 。然后，分别与 32-bit 的系统参数 FK_0, FK_1, FK_2, FK_3 异或，记为：

$$K_{-4} = MK_0 \oplus FK_0,$$

$$K_{-3} = MK_1 \oplus FK_1,$$

$$K_{-2} = MK_2 \oplus FK_2,$$

$$K_{-1} = MK_3 \oplus FK_3.$$

其中, $FK_0 = 0xA3B1BAC6$, $FK_1 = 0x56AA3350$, $FK_2 = 0x677D9197$, $FK_3 = 0xB27022DC$ 。

2. 生成轮密钥 $K_i (i = 0, \dots, 31)$:

$$K_i = K_{i-4} \oplus T'(K_{i-3} \oplus K_{i-2} \oplus K_{i-1} \oplus CK_i).$$

其中, 函数 T' 只需将加密算法中的函数 $T = L \circ S$ 中的线性变换 L 替换为

$$L'(B) = B \oplus (B \lll 13) \oplus (B \lll 23),$$

固定参数 $CK_i (i = 0, \dots, 31)$ 的具体值如下表所示:

表 2: CK参数表

00070E15	1C232A31	383F464D	545B6269
70777E85	8C939AA1	A8AFB6BD	C4CBD2D9
E0E7EEF5	FC030A11	181F262D	343B4249
50575E65	6C737A81	888F969D	A4ABB2B9
C0C7CED5	DCE3EAF1	F8FF060D	141B2229
30373E45	4C535A61	686F767D	848B9299
A0A7AEB5	BCC3CAD1	D8DFE6ED	F4FB0209
10171E25	2C333A41	484F565D	646B7279

2.1.1.3 解密算法

因采用广义Feistel结构, SM4算法具有加解密结构一致性。因此, 解密算法与加密算法一致, 仅需将参与第 i ($i = 1, \dots, 32$) 轮迭代运算的轮密钥由 K_{i-1} 替换为 K_{32-i} 即可。

2.1.2 SM3算法

SM3 算法是中国国家密码管理局于 2010 年发布的哈希算法, 属于国密标准的一部分。相较于其他传统哈希算法, SM3 在多个方面展现出显著优势: 哈希复杂度高、处理速度快、资源消耗更低; 其计算效率在大规模数据处理场景中表现尤为突出。

根据国家密码管理局发布的SM3总则[12], 相关符号及前置理论知识论述如下。

2.1.2.1 符号定义

表 3: SM3 符号定义表

符号	含义
$ABCDEFGH$	8个寄存器或他们值串联
$B^{(i)}$	第 <i>i</i> 个消息分组
CF	压缩函数
FF_i	布尔函数
GG_i	布尔函数
IV	初始值
P_0	压缩函数中的置换函数
P_1	消息扩展中的置换函数
T_j	常量
m	消息
m'	填充后的消息

2.1.2.2 常量与函数定义

(1) 初始值

$$IV = 7380166f\ 4914b2b9\ 172442d7\ da8a0600\ a96f30bc\ 163138aa\ e38dee4d\ b0fb0e4e$$

(2) 常量

$$T_j = \begin{cases} 79cc45190 & 79cc45190 \leq j \leq 15 \\ 7a879d8a16 & 7a879d8a16 \leq j \leq 63 \end{cases}$$

(3) 布尔函数

$$FF_j(X, Y, Z) = \begin{cases} X \oplus Y \oplus Z & 0 \leq j \leq 15 \\ (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z) & 16 \leq j \leq 63 \end{cases}$$

$$GG_j(X, Y, Z) = \begin{cases} X \oplus Y \oplus Z & 0 \leq j \leq 15 \\ (X \wedge Y) \vee (\neg X \wedge Z) & 16 \leq j \leq 63 \end{cases}$$

(4) 置换函数

$$P_0(X) = X \oplus (X \lll 9) \oplus (X \lll 17)$$

$$P_1(X) = X \oplus (X \lll 15) \oplus (X \lll 23)$$

2.1.2.3 算法描述

填充

SM3的消息扩展步骤是以512位的数据分组作为输入的。因此，我们需要在一开始就把数据长度填充至512位的倍数。数据填充规则和MD5一样，具体步骤如下：

1. 先填充一个“1”，后面加上k个“0”。其中k是满足 $(n+1+k) \bmod 512 = 448$ 的最小正整数。
2. 追加64位的数据长度

迭代压缩

将 m' 按512比特进行分组的到 $m' = B^{(0)}B^{(1)}\dots B^{(n-1)}$ ($n = (l + k + 65)/512$)

迭代方式如下：

$$V^{i+1} = CF(V^{(i)}, B^{(i)}) (0 \leq i \leq n - 1)$$

消息扩展

将分组 B^i 按照如下方法扩展成132个字 $W_0, W_1, \dots, W_{67}, W'_0, W'_1, \dots, W'_{63}$ ：

1. 将消息分组 B^i 划分为16个字 W_0, W_1, \dots, W_{15}
2. $W_j = P_1(W_{j-16} \oplus W_{j-9} \oplus (W_{j-3} \lll 15)) \oplus (W_{j-13} \lll 7) \oplus W_{j-6}$ ($16 \leq j \leq 67$)
3. $W'_j = W_j \oplus W_{j+4}$ ($0 \leq j \leq 63$)

压缩函数

令 A, B, C, D, E, F, G, H 为字寄存器， $SS1, SS2, TT1, TT2$ 为中间变量，压缩函数 $V^{i+1} = CF(V^{(i)}, B^{(i)})$, $0 \leq i \leq n - 1$ 。计算过程描述如下：

$$ABCDEFGH \leftarrow V^{(i)}$$

FOR $j = 0$ **TO** 63

```

 $SS1 \leftarrow ((A \lll 12) + E + (T_j \lll j)) \lll 7$ 
 $SS2 \leftarrow SS1 \oplus (A \lll 12)$ 
 $TT1 \leftarrow FF_j(A, B, C) + D + SS2 + W'_j$ 
 $TT2 \leftarrow GG_j(E, F, G) + H + SS1 + W_j$ 
 $D \leftarrow C$ 
 $C \leftarrow B \lll 9$ 
 $B \leftarrow A$ 
 $A \leftarrow TT1$ 
 $H \leftarrow G$ 
 $G \leftarrow F \lll 19$ 
 $F \leftarrow E$ 
 $E \leftarrow P_0(TT2)$ 
ENDFOR

```

$$V^{(i+1)} \leftarrow ABCDEFGH \oplus V^{(i)}$$

输出杂凑值

$$ABCDEFGH \leftarrow V^{(n)}$$

输出256比特的杂凑值 $y = ABCDEFGH$ 。

2.1.3 NTRU算法

NTRU是一种基于环与格的公钥密码系统，NTRU是Hoffstem、Pipher和Silverman在1996年提出的一种新的公钥密码体制。该体制是建立在多项式环的基础之上的，其安全性基于格上最短向量问题。NTRU密钥短且容易产生，算法运算速度快，所需存储空间小，与目前广泛使用的公钥密码系统RSA及椭圆曲线密码系统相比较，在安全要求相同的情况下，NTRU的特征简洁，密钥生成不复杂，运算速度快并且需要的存储空间小，所以目前基于格上困难问题建立密码体制的算法中，NTRU更加简便高效[21]。但NTRU的原始方案的安全性一直没有得到严格的证明。2011年，Stehle，Steinfeld在理想格上基于R-LWE问题构造了选择明文攻击安全的NTRU加密体制。2012年，Steinfeld等人在理想格上提出了选择密文攻击安全的NTRU加密体制。

下面介绍NTRU的加解密方案的整个过程，相关参数定义与取值如下。

2.1.3.1 相关参数定义与取值

表 4: NTRU相关参数和取值

符号	定义	取值
n	次数参数	743
q	正整数大模数	64
p	小的奇素数或多项式	3
d	限制非零系数的个数	72
$f(x)$	n 次整系数多项式	$x^n - 1$

2.1.3.2 密钥生成阶段

1. 随机选取 $\mathbb{Z}[x]$ 上的多项式 $f(x)$, $f(x)$ 的次数小于 n , 有 $d + 1$ 个系数为 1, d 个系数为 -1, 其他系数为 0。
2. 检验是否 $f(x)$ 在环 $\mathbb{Z}_q[x]/f(x)$ 上可逆, 如可逆则计算逆元 $f_q^{-1}(x)$ 。
3. 检验是否 $f(x)$ 在环 $\mathbb{Z}_p[x]/f(x)$ 上可逆, 如可逆则计算逆元 $f_p^{-1}(x)$ 。
4. 重复进行上述步骤, 直到计算出 $(f_q^{-1}(x), f_p^{-1}(x))$ 。
5. 随机选取 $\mathbb{Z}[x]$ 上的多项式 $g(x)$, $g(x)$ 的次数小于 n , 有 $d + 1$ 个系数为 1, d 个系数为 -1, 其他系数为 0。
6. 公钥 $h(x) = pg(x)f_q^{-1}(x) \bmod q \bmod f(x)$, 是环 $\mathbb{Z}_q[x]/f(x)$ 上的多项式。私钥是 $\{f(x), g(x)\}$, 而 $p = 3$ 意味着 $f(x)$ 和 $g(x)$ 既是环 $\mathbb{Z}_p[x]/f(x)$ 上的多项式, 又是环 $\mathbb{Z}_q[x]/f(x)$ 上的多项式。

2.1.3.3 加密阶段

明文空间为 $\{-1, 0, 1\}^n$ 。也将明文空间表示为次数小于 n , 每个系数属于 $\{-1, 0, 1\}$ 的多项式的全体。而 $(p, f(x)) = (3, x^n - 1)$ 意味着明文空间就是环 $\mathbb{Z}_p[x]/f(x)$ 。

对于明文 $m(x)$, 进行如下的操作:

1. 在环 $\mathbb{Z}_p[x]/f(x)$ 上随机选取多项式 $r(x)$, $r(x)$ 作为随机化因子。
2. 计算密文 $c(x) = r(x)h(x) + m(x) \bmod q \bmod f(x)$ 。

2.1.3.4 解密阶段

1. 计算 $d(x) = c(x)f(x) \bmod q \bmod f(x)$ 。
2. 计算 $m^*(x) = d(x)f_p^{-1}(x) \bmod p \bmod f(x)$ 。

注意到 $d(x) = \text{pr}(x)g(x) + m(x)f(x) \bmod q \bmod f(x)$, 如果 $\text{pr}(x)g(x) + m(x)f(x) \bmod f(x)$ 的每个系数在区间 $(\frac{-q}{2}, \frac{q}{2})$ 内, 则 $d(x) = \text{pr}(x)g(x) + m(x)f(x) \bmod f(x)$, 模 q 运算省略掉了。

因此有 $m^*(x) = \text{pr}(x)g(x)f_p^{-1}(x) + m(x) \bmod p \bmod f(x) = m(x)$ 。

2.2 方案实现

在过去的 research 中, 有研究将不对称加密算法SM2和对称加密算法SM4相结合, 对对称加密算法的密钥和明文信息进行加密, 实现双重加密效果。实验结果表明, 该设计的系统提高了信息传输和密钥共享的安全性[14]。

本部分将首先介绍利用SM3和SM4实现的Feistel分组密码加密结构, 我们将命名为LBC(Large Block cipher), 然后讲述如何将NTRU应用到我们的加密方案中以增强密钥的保密性, 之后会对我们如何对明文分组以及工作模式如何处理分组进行说明, 最后介绍对密文头部的处理

2.2.1 大分组分组密码LBC实现

2.2.1.1 加密算法

加密算法如下图所示

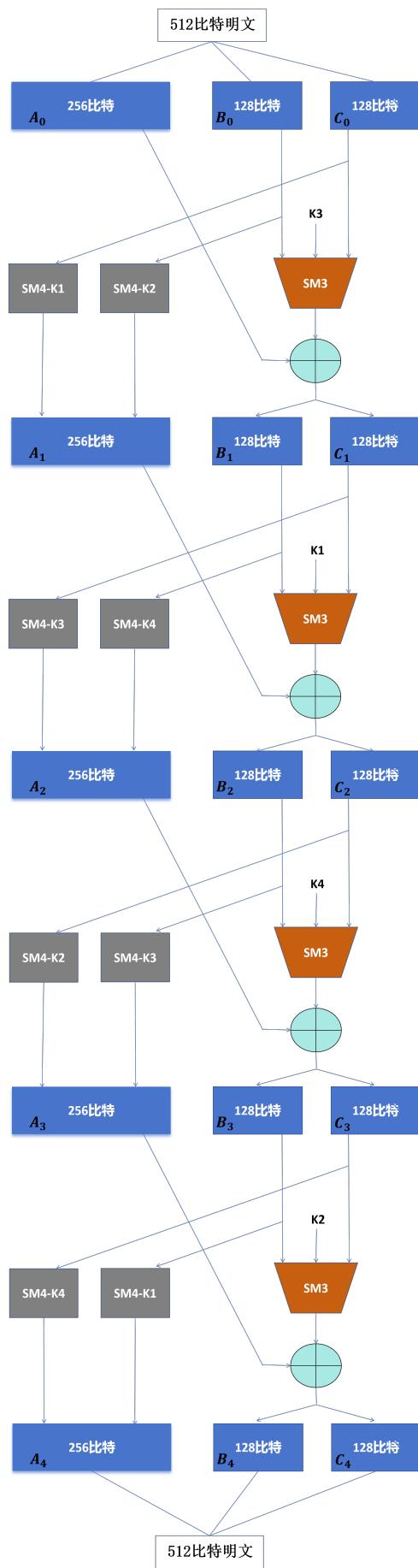


图 2: 加密结构

具体实现有如下

- (1) 输入的512-bit明文，按256-bit、128-bit、128-bit划分为三个部分，记为 (A_0, B_0, C_0) 。
- (2) 进行四轮迭代，每轮迭代会用到3个128bit的密钥记为 $K_0 K_1 K_2$ ，每轮用到的密钥组合各不相同。
- (3) 每轮加密开始时，运算如下：

$$A_{i+1} = SM4(B_i, K_2) \parallel SM4(C_i, K_1)$$

$$B_{i+1} \parallel C_{i+1} = A_i \oplus SM3(B_i \parallel K_3 \parallel C_i)$$

- (4) 最后得到512比特密文 $A_4 \parallel B_4 \parallel C_4$ 。

2.2.1.2 密钥分配

在我们加密算法中，接受的密钥长度可在128、256、384、512-bit之中，最终用在每轮算法的密钥有4个记为 K_1, K_2, K_3, K_4 ，都长128-bit，因此针对不同的输入密钥 K_0 长度会有不同分配方式，具体分配如下表：

表 5: 密钥分配

输入密钥长度	分配方式
128-bit	$K_1 = K_2 = K_3 = K_4 = K_0$
256-bit	$K_1 \parallel K_2 = K_0, K_3 = K_1, K_4 = K_2$
384-bit	$K_1 \parallel K_2 \parallel K_3 = K_0, K_4 = K_1 \oplus K_2 \oplus K_3$
512-bit	$K_1 \parallel K_2 \parallel K_3 \parallel K_4 = K_0$

2.2.1.3 解密算法

结合feistel结构特点以及用到的SM3，SM4算法给出如下一轮解密方法，对密文执行4轮即可解密出明文：

- (1) 输入的512-bit密文，按256-bit、128-bit、128-bit划分为三个部分，记为 (A_i, B_i, C_i) 。
- (2) 用到的三个128-bit密钥分别记为 K_0, K_1, K_2 ，利用密钥 K_0 和 A_i 的左128-bit输入SM4解密得到 C_{i-1} ，同理利用 K_2 和 A_i 的右128-bit输入SM4解密得到 B_{i-1} 。
- (3) 用得到的 $B_{i-1} \parallel K_2 \parallel C_{i-1}$ 输入到SM3中得到杂凑值，然后将杂凑值与 $B_i \parallel C_i$ 异或得到 A_{i-1} 。
- (4) 最后得到的 $A_{i-1} \parallel B_{i-1} \parallel C_{i-1}$ 就是解密一轮的结果。

2.2.2 数字信封

为了在物理世界中托管数据，可以将其存储在一个密封且防篡改的信封中，这样信封可以被打开，但一旦打开就无法重新密封。在本文中，数字信封的概念是一种对物理世界中信封的数字模拟。

数字信封允许爱丽丝以一种方式将数字数据提供给第三方，使得第三方只有以下两种可能的操作：

- 他可以在不需要爱丽丝进一步操作的情况下访问数据。
- 或者，他可以放弃访问数据的权利，在这种情况下，他能够向爱丽丝证明他没有并且不能（再）访问数据。

直观上，仅仅使用加密技术无法实现这种效果。爱丽丝可以加密数据并将密文发送给第三方。但如果她不同时发送密钥，那么没有爱丽丝的进一步合作，第三方无法解密数据。如果她在发送数据的同时发送密钥，那么第三方就无法证明他没有解密数据。即使他将密文“归还”给爱丽丝，她也无法保证他没有解密另一个副本[16]。

在我们的软件中，两方用户对文件消息加密共享时，一个用户进行加密，算法会自动为用户生成随机密钥，最终密钥和密文一起生成，如果直接通过和密文一样的不可靠信道传播密钥，可能导致密钥同密文被攻击者窃取从而导致密钥泄露：

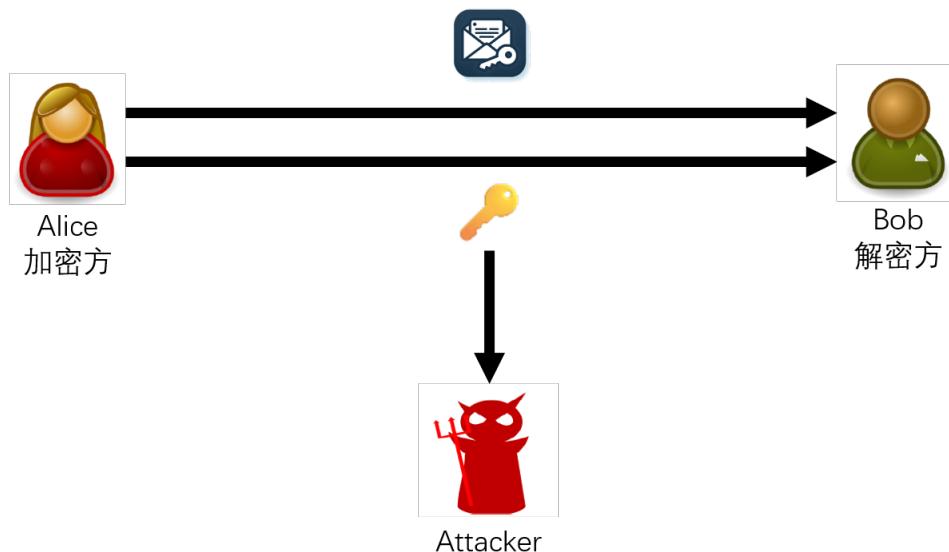


图 3: 一般情况密钥传播

为了避免对称密钥泄露，我们采用了数字信封的方式，两方用户需提前商议好NTRU的公私钥，每次加密用户加密时输入NTRU公钥，对随机生成的分组密码密钥进一步加密，达到密封效果，此时加密后的密钥就可以直接和密文一起发送给另一方，另一方用提前商议好的NTRU私钥再对信封解密得到分组密钥的明文，再用分组密钥对消息密文解密即可得到最终的消息明文。

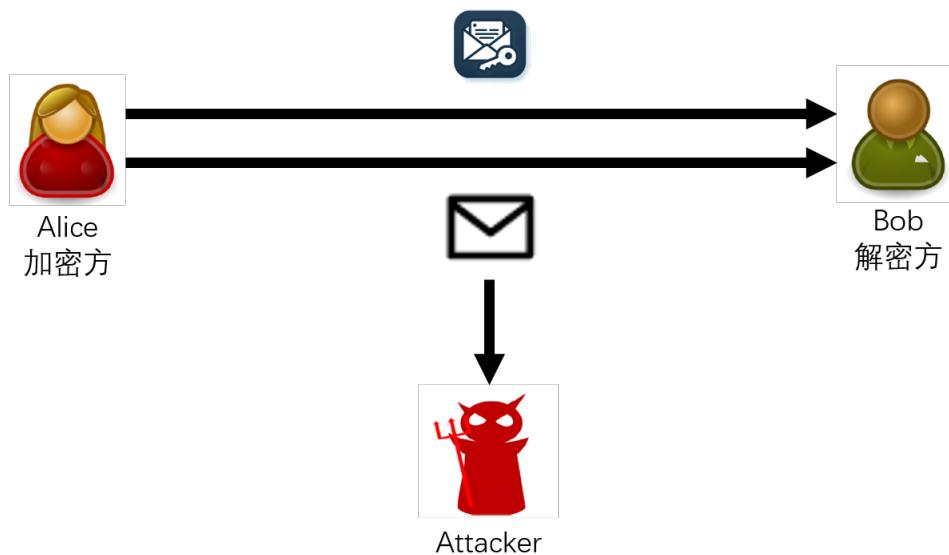


图 4: 数字信封的密钥传播

2.2.3 明文分组

由于我们的软件核心算法是LBC大分组分组密码，因此对于长明文就要进行分组处理，每个分组最大长度为512-bit，对于三种明文长度情况我们作出如下分组处理，记明文长度为 l ，单位为bit：

表 6: 分组策略

明文长度	分组方式
$l = n * 512 (n \geq 1)$	正常分组
$l > 512, l \bmod 512 \neq 0$	密文偷垒
$l < 512$	PKCS#7填充

密文偷垒

密文偷垒指的是当最后一组明文不满组时，将倒数第二组加密的密文挪用给最后一组明文使用，来完成填充的过程，针对CBC和ECB模式有不同的密文偷垒方法，如下图所示：

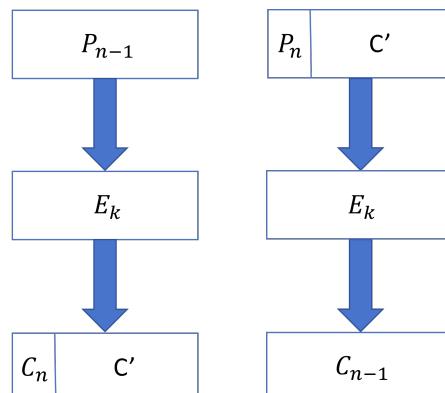


图 5: ECB加密

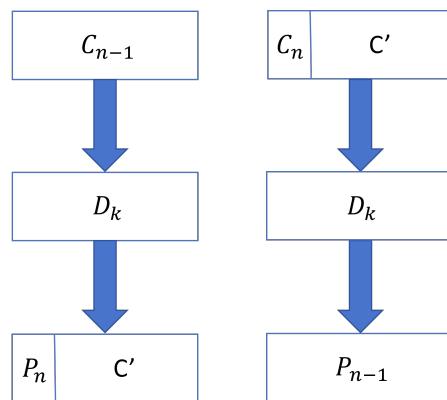


图 6: ECB解密

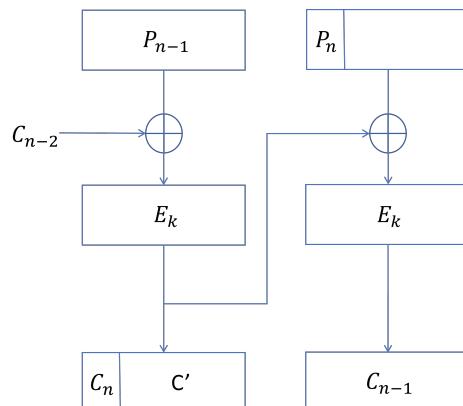


图 7: CBC加密

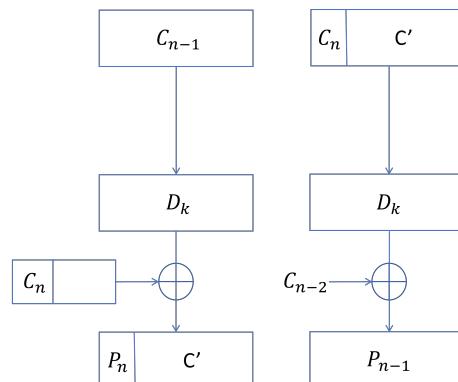


图 8: CBC解密

PKCS#7填充

密文偷垒的方法在明文长度不满一个分组时便失去作用，因为将不存在倒数第二个分组，因此这种情况下只能通过填充实现分组，PKCS#7是当下各大加密算法都遵循的填充算法，且 OpenSSL 加密算法默认填充算法就是 PKCS#7。其具体的填充方式就是当消息长度不足一组时，统计缺少的字节数，缺少多少个字节就在消息末尾补充多少字节的缺少字节数[17]。

2.2.4 密文头部处理

为了使解密能顺利进行，在对文件加密完成之后，需要对文件头部添加一些信息，包括加密模式，初始IV等，因此我们定义如下密文头：

- 1) 第一字节作为固定字符，表示版本号，置为0x01
- 2) 第二字节作为加密模式，CBC加密置为0，ECB加密置为1
- 3) 之后往后的64字节存储随机生成的初始IV，这里虽然只有CBC需要初始IV，但ECB模式下也将IV添加了进去

如图：



图 9: 密文结构

在解密的时候就不需要用户得知工作模式，因为接口算法能通过拆解密文头找到工作模式标识从而确定是CBC还是ECB，在CBC情况下，便把之后的64字节作为IV进行接下来的解密，否则ECB模式会直接跳过这64字节寻找密文本体。

2.2.5 软件目录结构

最终我们得到的软件目录结构如下

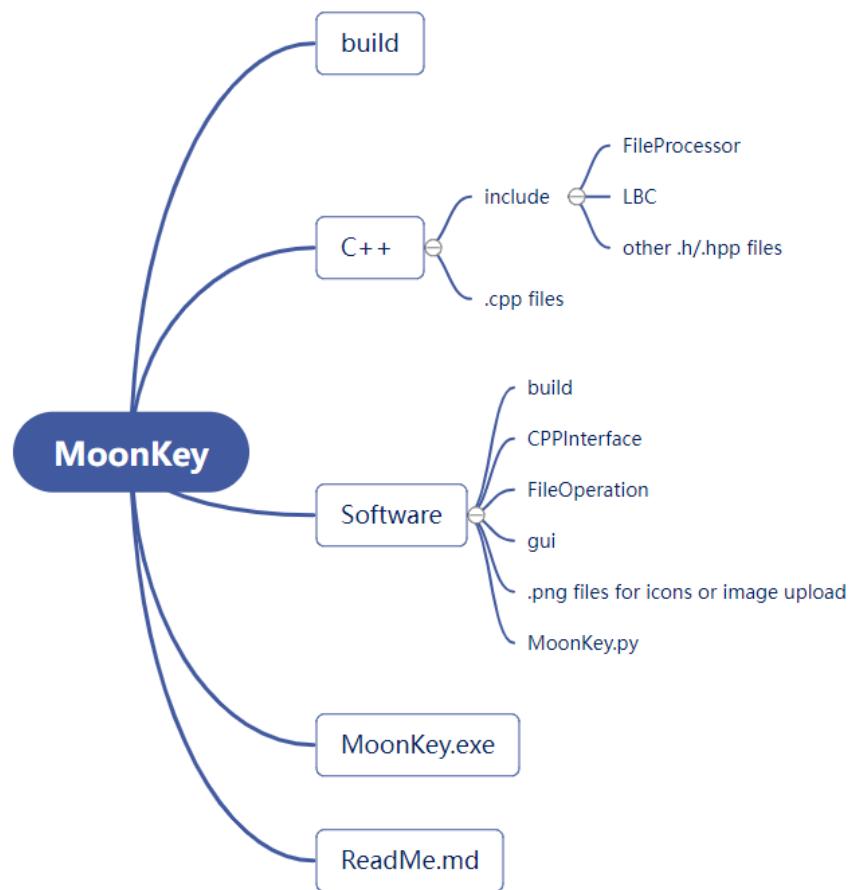


图 10: 目录结构

- MoonKey/C++/include/FileProcessor 用于存放处理文件的头文件
- MoonKey/C++/include/LBC 用于存放用于加密、解密的头文件
- MoonKey/Software/CPPInterface 用于提供 python 中 C++ 接口的动态库
- MoonKey/FileOperation 用于处理文件
- MoonKey/gui 用于提供用户界面

3 系统测试与结果

3.1 测试方案

我们分别在算法正确性，功能完备性和性能高效性三方面进行了测试方案的安排，如下图所示：

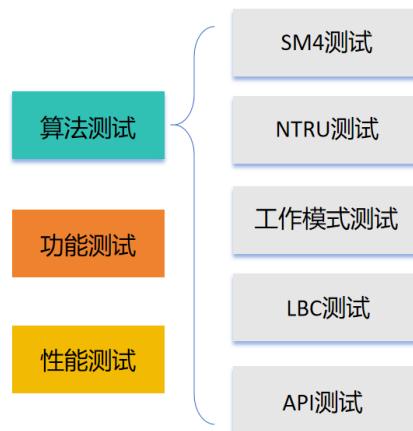


图 11: 测试方案

3.2 算法测试

我们对实现的一些密码算法组件进行了正确性测试，主要通过解密加密后的密文得到明文与原明文对照，若一致能说明加解密算法执行起来的互逆是没问题的，我们在C++算法源码上写入了算法测试部分，对每一部分的测试设置了测试id，运行main函数后可根据id进行测试选项：

表 7: 测试id表

test id	测试项目
0	NTRU测试
1	SM4测试
2	LBC测试
3	CBC工作模式测试
4	ECB工作模式测试
5	API测试
-1	退出测试

由于在我们的算法中，有两种工作模式和4种密钥选择，对长明文分组处理时又有三种策略，因此如果将全部测试结果列出会导致过多，我们有针对性地选择几种策略在下面进行了展示，测试细节和结果如下

3.2.1 NTRU测试

在NTRU算法测试中，我们的参数选择为 $N = 743, p = 3, q = 64, df = 15, dg = 12, dr = 5$ ，这也是我们最终应用到软件当中的NTRU参数。我们将明文设为”hello world”，密钥随机生成，进行一次NTRU的加解密，执行测试代码，将测试id输入为0，得到如下结果：


```

* ===== *
||Test
||0. NTRU
||1. SM4
||2. LBC
||3. LBC-CBC
||4. LBC-ECB
||5. API
||-1. exit
* ===== *
* input test_id: 2
* LBC_test
Randomly gen_mainenerated plaintext:
5fbce60c      6a8c5238      800b58d7      b80ecc5d
224fb567      a15fce1       8347be9b      7f1b6d04
89b30226      c81ca3f6       21e8ab3       917c90a8
6b0a8d7e      fe018e98       64796680      27bb08ba

Randomly gen_mainenerated keys:
ef70fc66      ed92fc35      3d82b0a4      ecfe0236
d41d36fd      53dd9950      169a7fa2      7322c922
7451c885      5738d180      f7a2a541      beb07eba
16146a85      e4f900d2      c755f1c6      a10003ac

after encrypt, ciphertext:
e52e3fc1      dc59e854      d22ef6b1      942beefc
ac6de6bd      304ff9df      c2cf536       59c60cf
ed80cd6b      d9817ea3      9613b6aa      714864d0
744015fa      7e38de22      fd59a6ab      4ea22417

after decrypt, plaintext:
5fbce60c      6a8c5238      800b58d7      b80ecc5d
224fb567      a15fce1       8347be9b      7f1b6d04
89b30226      c81ca3f6       21e8ab3       917c90a8
6b0a8d7e      fe018e98       64796680      27bb08ba

```

图 14: LBC算法测试

3.2.4 CBC工作模式测试

随机生成密钥和明文，密钥为512-bit，明文为1056-bit，设置的不是512bit的整数倍，以此检验密文偷垒是否正常，得到如下结果：

```

* ===== *
||Test
||0. NTRU
||1. SM4
||2. LBC
||3. LBC-CBC
||4. LBC-ECB
||5. API
||-1. exit
* ===== *
* input test_id: 3
* LBC-CBC_test
m:
b66d2364      8c6988cd      90821bb5      6a955a0d     faf262c6      7b51c97f      b033e6e3      baeb84c3
568daffd      9d3f1a6       8c7b2e84      31a6d70f      63c7ce19      d8843586      7a708c75      7f4bfbae
e42655a0      b44e3a77      d6aef4c7      a29acca3      4b86b84a      22007393      49f522e6      35eab7f9
81561e59      cf105124      ececc2c90      96626bfc      40d52f71      ca05e94d      8c8aa180      7d1dc8c9
6b87ea4e
k:
3ccc8545      a376a353      672fe488      ec56d790      593389bb      c94443ba      6c062493      91f73419
9f7a87b       6ddf4438      ed932f21      931a8869      174b43f5      87adec99      b2449ac      91da388
c:
66db247       700dd872      7b359b18      8f531478      ad6ea4df      37bf1af       f16ebd25      70996ff7
19760d2       cf76358f      310d5eb8      f0030407      f20db904      b9255720      ef4a90ec      8004ab07
b67f4fd1      7184db7d      8c91e241      1f90deb5      380461ae      47257243      358cb321      e9fb9bd2
48142862      9833883     8a67d928      d1d3027f      ae3656b5      73748a55      156aecc4      7b920005
b57c9efc
m = Dec(c):
b66d2364      8c6988cd      90821bb5      6a955a0d     faf262c6      7b51c97f      b033e6e3      baeb84c3
568daffd      9d3f1a6       8c7b2e84      31a6d70f      63c7ce19      d8843586      7a708c75      7f4bfbae
e42655a0      b44e3a77      d6aef4c7      a29acca3      4b86b84a      22007393      49f522e6      35eab7f9
81561e59      cf105124      ececc2c90      96626bfc      40d52f71      ca05e94d      8c8aa180      7d1dc8c9
6b87ea4e

```

图 15: CBC工作模式测试

3.2.5 ECB工作模式测试

随机生成密钥和明文，密钥为512-bit，同样明文为1056比特，得到如下结果：

m	k	c	m	k	c	m	k	c
f89cd4a0	12daf7d5	14167a4c	f89cd4a0	12daf7d5	14167a4c	f89cd4a0	12daf7d5	14167a4c
6d1a7927	c30761ca	c45a3b82	6d1a7927	c30761ca	c45a3b82	6d1a7927	c30761ca	c45a3b82
e9c03b9	72e8c2ca	acec3628	e9c03b9	72e8c2ca	acec3628	e9c03b9	72e8c2ca	acec3628
921bdd50	83d840e1	c6f48c99	921bdd50	83d840e1	c6f48c99	921bdd50	83d840e1	c6f48c99
327c86a4			327c86a4			327c86a4		
k:			k:			k:		
798d5c55	fa8569a2	9aac4cb3	798d5c55	fa8569a2	9aac4cb3	798d5c55	fa8569a2	9aac4cb3
8ea51fc8	b9c5291a	84b51c69	8ea51fc8	b9c5291a	84b51c69	8ea51fc8	b9c5291a	84b51c69
c:			c:			c:		
c621acd2	d97eala5	ec45d03d	c621acd2	d97eala5	ec45d03d	c621acd2	d97eala5	ec45d03d
448720a1	959eaf49	253f8cb	448720a1	959eaf49	253f8cb	448720a1	959eaf49	253f8cb
705c4677	b82ca635	fb0d74a1	705c4677	b82ca635	fb0d74a1	705c4677	b82ca635	fb0d74a1
7e5b1db8	90613d8b	3c964130	7e5b1db8	90613d8b	3c964130	7e5b1db8	90613d8b	3c964130
5b82d048			5b82d048			5b82d048		
m = Dec(c):			m = Dec(c):			m = Dec(c):		
f89cd4a0	12daf7d5	14167a4c	f89cd4a0	12daf7d5	14167a4c	f89cd4a0	12daf7d5	14167a4c
6d1a7927	c30761ca	c45a3b82	6d1a7927	c30761ca	c45a3b82	6d1a7927	c30761ca	c45a3b82
e9c03b9	72e8c2ca	acec3628	e9c03b9	72e8c2ca	acec3628	e9c03b9	72e8c2ca	acec3628
921bdd50	83d840e1	c6f48c99	921bdd50	83d840e1	c6f48c99	921bdd50	83d840e1	c6f48c99
327c86a4			327c86a4			327c86a4		

图 16: ECB工作模式测试

3.2.6 API测试

这里测试的API是用于给软件提供的接口，相当于对前面那些密码组件测试的综合，我们依然选择随机生成512-bit的LBC密钥和1056-bit的明文，工作模式分别选择CBC和ECB进行了测试，对于NTRU部分密钥也是随机生成的，测试结果如下：

m	k	c	m	k	c	m	k	c
8719f38b	91daea914	4l-65ffe2	8719f38b	91daea914	4l-65ffe2	8719f38b	91daea914	4l-65ffe2
4799e229	be0d75b5	773b167d	4799e229	be0d75b5	773b167d	4799e229	be0d75b5	773b167d
8b5c87ca	be20d56c	ebf079cb	8b5c87ca	be20d56c	ebf079cb	8b5c87ca	be20d56c	ebf079cb
e8f469c3	a46b2bbd	c43436fb	e8f469c3	a46b2bbd	c43436fb	e8f469c3	a46b2bbd	c43436fb
6eabf417			6eabf417			6eabf417		
k:			k:			k:		
F0pdXvGFn8WvnaJGp3ljqwCe0hFH=j+foaw3ta+R=e39jvLxDzTB5k0mcQEbmXuNerkew92S0vhFHTyD8Bk1idumLPhQxVb2zr9k=h=Hn+OxivM570Wjv4NPHRovqLZ8r5fksw8Rdl.BhomfJKaclolkvB76m=8czTsqxxr1+s8VoG6ZRuWceefPUflNrxLxuYVggeh0BvR0RTu8W0QycD3SHPx8vF95tGh9Xb1re+=+Ydyv5fGrxv9Af081smQoGCapbuwPDTD+JHEqe==lry2YoJv9BuHNUMeVhGyWk7PhXoFvc8mBwGoexcuwoI=OInhnfzve1bn29vbj2Lcp56yoAhPw29knIsD28shlkXA1ja7K0L5shekUP1SKXRSo4r4ukfNgRj30n7txrPACCBnKueT3=x30Co+JelPbZEYCKU5w2An4+Q3o0uIglXLx2rwFkg5Sr+0kpZbCnFnTneJ7uo8iEY63NFh1jpM:R15oYw10Pr3UemDI310+aBjMnw=C35+o4yAaPzWkDzSU2W0Hd5dqzvNwajwHek1lk0epb8k0ZvIu0-D8hy=LrpzbE=ll1nnB1nTfL==20mGwCgp0s1zeu0wluUzfv3wnNwqYy3CnUs5hJo3a3nkhx8zfrH0qZy6gHSVbfMtMKY106QxCrs1kwFr2mbgHSzeMMB06D7Rz2V2UxX15BLgCDEds1Jx0BExcef1Jx1Acjn1zMFKr=jZMguJ3c2Fx5ohzCwzcmhQJbd7x2C-UdsjGdknRHjxMw44jhjzcsH5xWdf85c1hIO=uJxTWDMMBZyuuft1HLfYNX+6zFXxxFMFdPeyb0uZ3R7Au7z+5sC2Pc3oANW1Ej0bjRZNGheoSsCaNCJ81cPbaTjzVf6aeNake7wIJU+8hko9ShBw8TtakxtCw=j801PdnA1NMpr5mJn26+b9Qk00L5NsawAaOLCOhvugCw==pNTNgaaCjg=F0n4sNEVgza76k7sed0s0wYGR0cBnAb+iaePyCcXjQHvYy0Fj0VFFJx1KLAPGIm2=Z69gdAomsXQkPEWGL4g+ihy78vhbj5ehVkhJf4nLQ9x1Go1FaPB9e+46BvLfZbr+uQnw2UIJ480CH0CQqhBpzoBkor3sx4Wt4lxwN6Ja286sKehal0+8P6dGGZelMgj1oC0t43ryh5pdajFzKH3os+dzckvG0PNrrP8Cmfwldfdg6vf!								
c:			c:			c:		
100	23dc209c	a0ff78	467uf99b	47ce96e	2bfe0de8	cbdeb1ea	4599ec85	
95f387e9	20bb6f2	edf1e0	e598e41	3c9b92a	f355961	3fb6c244	58ffcf367	
5b82d048	b7b6d5	d79b074	c6c8b4	5259d4	3e3485d5	1a02738a	262e684	
27ba70ad	68c5f6b	6938492d	b9948edf	5385b13	1fc43f3e4	74234cd	91aa9925	
14b43278	483b3082	3abde25	37562cf6	95dc920	fa04e39	8344ec4d	a721dd07	
9e7f1461	91a1e9b9	2f359a7	33ff6f1b	b4776d9	11497282	d76dd315	4ec7f499	
94368cfb	2eca5466							
m = Dec(c):			m = Dec(c):			m = Dec(c):		
8719f38b	91daea914	4c65ffe2	8719f38b	91daea914	4c65ffe2	8719f38b	91daea914	4c65ffe2
4799e229	be0d75b5	773b167d	4799e229	be0d75b5	773b167d	4799e229	be0d75b5	773b167d
8b5c87ca	be20d56c	ebf079cb	8b5c87ca	be20d56c	ebf079cb	8b5c87ca	be20d56c	ebf079cb
e8f469c3	a46b2bbd	c43436fb	e8f469c3	a46b2bbd	c43436fb	e8f469c3	a46b2bbd	c43436fb
6eabf417			6eabf417			6eabf417		

图 17: API-CBC测试

```

=====
|| Test
|| 10. NTRU
|| 11. SMU
|| 12. LBC
|| 13. LBC-CBC
|| 14. LBC-ECB
|| 15. API
|| -1. exit
=====
* =====*
* input test_id: 5
* api_test
m:
f9a6402e 36a40791 294189a5 9da9e9f5d a863b197 4f3347d1 de64452c da9df27
76b972cd ebb45aea 10800035 92a5e00c 4c0b2ac9 f0609b25 b670c0eb 3dd888
742226c3 dfeef3e 4c79865c bf2c337 8e10badd f8cf6f2 3e57e1d1 8af5a03
d26f65fa d9b417d9 ff905298 dbdac0d 2e21e33c 2b9de05d dfbfb929e aa13d731
a98c1dfa
k:
6J4gdBMF6fpCdc1pmh-FBUb-BSU3r0eYhbHLiOASt5Ne0zL1h4299NP1daorSw0LzPG1aSMEQ13EhawNwGvmeA0rEMaPzPENy9K29xDx2cIILtLwV2zcfxegwJeaIIIdIDzszkvCtzgC0HEwDXPl4fzxUuqV12D3rUZuGUv16Z21aKf
0Z6bV5fV8-MzFVf1f6f1f6D14DUXk1p-nTl2z94CwzgR-MsM4z5-1aFVd4HTAT1wNP02d1L-AVEU6gnM1W1uD2798FTaxkrpghM3pugNltj23hdCU3Btewzv9V1L7Fz1V1AP4h4f531Uj3hYvT7UlkcuPchM
nLkM0bz2D0WiaelVWnhHuX8mwo8H+3,pv508CU4iM0f5TC1f715rx02ss0hRSXC0298+33K1vcxyyDhLTc0mAd0x4c11d10F8p=0tBEv062fJBdjtL72L1=pgumlIbby7zyZkR=08CD8cZSM0spku5Dh1hHW6hSzEWl=ChqjHV0mAc5=x
2D4t-wtMgChetyCn3eUQMP1p7OnE9kws0Ckv095gj5tGH1JRMyrn8qodIgmurBmVYQUsOF1d3j1jYsFFZnPOHCl7vu70Cekig0kcc6s+B5SC82PxWe9zEengqeDTgCK8bf+rt19xu12xHVdXJGm19bxF0Y5kiPTZxfl2Wl1gE19jSMqK0
PgRR8qfIT2y_3LKO1878fm1ly01X80jv55p8s+yhRxy4rA1V1LLt0cx9+afPSQMLMo9+jh70k4ddYhY+r+pxFCGaZKw19SUEMLpi9csGaq9p7g9sdlyXr5xxLWj1q18x-DaSe6sLVg21wBHXMj0gGvWYLY1J04T5xg1W4KLVB9shaxqQVs
BGJnrs21QokkkG9Tm751aiw1h6bT799hblrxp13bfk1AW084Jxtw5pAno4QaQ6n4SLGrNaas6fSaYCHfgrFNE6Nf7VHt+sUy3GwMPpxeg31BDxxuun0t20MDeFpVfir dovg2uUT3qyRy90Cf0p197c8Yu+Cdg+s+L2s1ZNubTwRULQet+g8uKs
X0jCkphd4y5yjnDA=xgdt7yv9rAj1Y1lxHjsk6w=i16RhRLN188TA=GddAxM2=y=R+u5k1rfxP052=eDNMsP2BjCDSJuDpOrEfyn0p0m0hupPmUevHs=tw5wXh1lHjBsx1s4dj1Efipu=cx0hgwylfFqzv4j+tVVB7r7pQFc77ZT
mAC7ocCA7NSql5sVnrpxB1+jr+5new420sU124rBwdxYex1Txka3Re7QYc68S2gAyhsedhG09=x54B8tHsKVSLytMMU4k14uWdo/4XLFn1JFSeQERDH4tx1Li2k5jMASyK0f4t7s1qc3N2jtdq=ExJ1dLdq8jTlrxJYOn2SvwFMA5
3yH=Fv9HrCBAs5yjW2oM5UpvD9yvz7RQjw=Nb6m40lyHs9pGkb69!
c:
101 bf03ed7b 98d54376 367d85f4 39d78f77 e0deee94 3afab110 a660e69
21d19af0d 6a391b0 7ff4490b4 917c90c3 c6b18ed4 6f77ab4f ef01c208 dbb26932
1efffc84 70eaccf b1f0a1ff a0647862 62352b7 652e821e c7ba619c 92322bf1
c807da4 6dd67ccf 8a53d566 ec72288 498b4e96 fa00febe fb5711f0 24ec91f7
e6128ba8 a6014ad1 4b62e6ac b6c5eb6c 57769ba7 fd44cf77 9228999c cf14ed6
584b417a 89389306 74289cb1 1fcf801 5bede51 97db637 873863c7 22e6d5a2
a4222f5c5 98c521f3
m = Dec(c);
f9a6402e 36a40791 294189a5 9da9e9f5d a863b197 4f3347d1 de64452c da9df27
76b972cd ebb45aea 10800035 92a5e00c 4c0b2ac9 f0609b25 b670c0eb 3dd888
742226c3 dfeef3e 4c79865c bf2c337 8e10badd f8cf6f2 3e57e1d1 8af5a03
d26f65fa d9b417d9 ff905298 dbdac0d 2e21e33c 2b9de05d dfbfb929e aa13d731
a98c1dfa

```

图 18: API-ECB 测试

3.3 功能测试

在这里将介绍我们软件最终成品的一些功能与用法，包括对文件的加密，对文件的解密，NTRU密钥生成三个主要功能

密钥生成

首先介绍NTRU密钥生成，由于我们软件主要用于文件加密，默认加解密双方是提前准备好NTRU公私钥的，因此我们在软件UI里没有直接添加ntru密钥生成选项，但为了方便演示，我们在软件文件夹/MoonKey/Software/CPPIInterface里提供了一个生成ntru密钥的python脚本，运行该脚本后，我们将得到如下的结果：

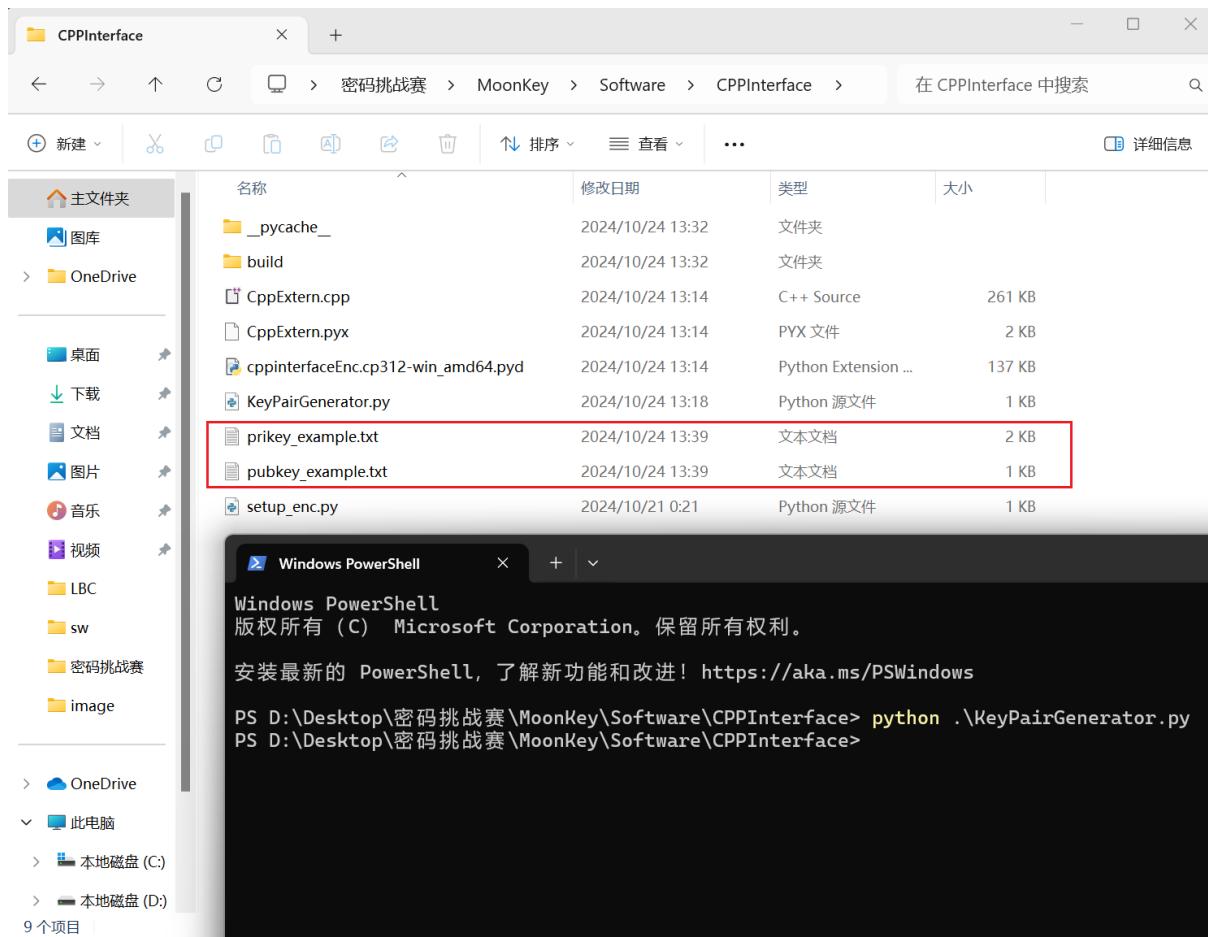


图 19: NTRU密钥生成

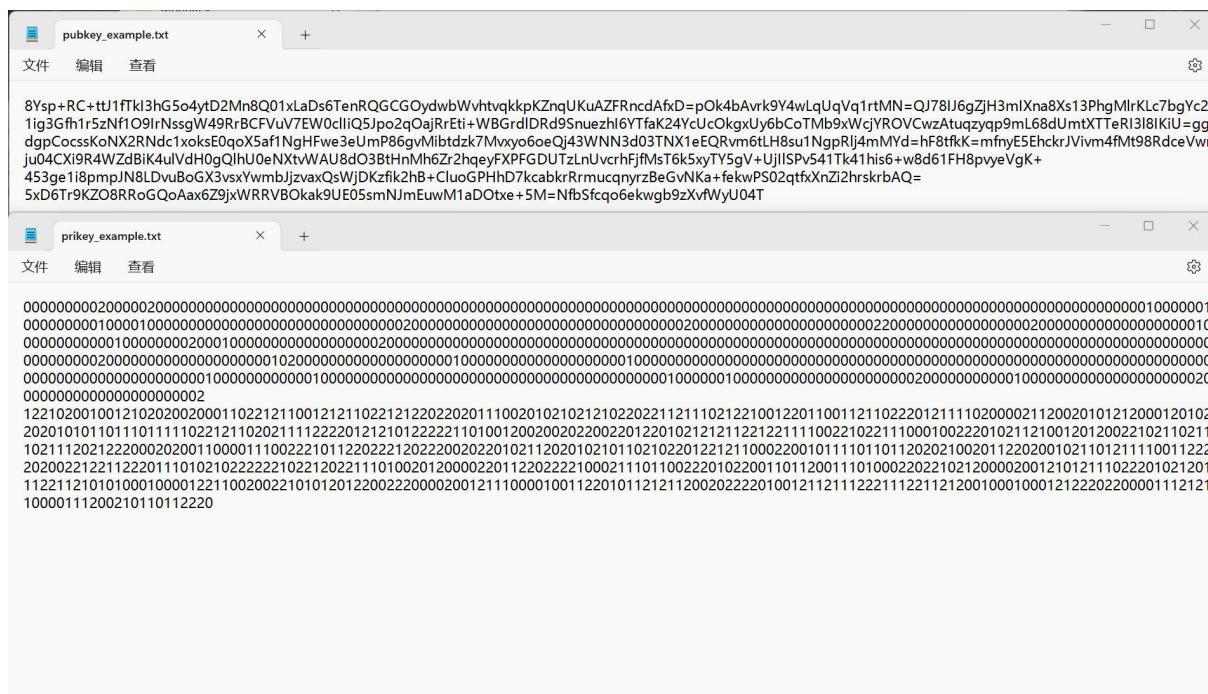


图 20: NTRU密钥展示

通过这种方法可以得到适合我们算法的NTRU公私钥，接下来我们将利用这一对公私钥演示对文件的加解密

文件加密

首先说明一下文件格式：

- **.lenc**: 加密文件，如生成的example_cbc.lenc文件即为某文件类型名为example的文件经过CBC工作模式加密后生成的密文文件。需注意，为避免泄露文件头字节，密文文件名中不会给出明文文件类型
- **.lkey**: 密钥文件，如生成的example_128bit.lkey文件即为某文件类型名为example的文件经过128bit密钥加密后生成的密钥文件。需注意，为避免泄露文件头字节，密钥文件名中不会给出明文文件类型
- **.ldec**: 解密文件类型，如生成的example_dec.ldec文件即为某文件类型名为example的文件解密后生成的密文文件。需注意，软件给出部分文件自动识别的功能，即恢复文件若属于下列文件类型，则会直接恢复该类型的后缀名，若不属于，则需要解密方自己修改后缀名恢复原文件。
- 可恢复文件类型：.jpeg, .png, .gif, .pdf, .bmp, .tiff, .rar, .elf, .ps, .mp3, .mp4, .mpeg, .ogg

接下来演示加密，以下是软件的初始UI：

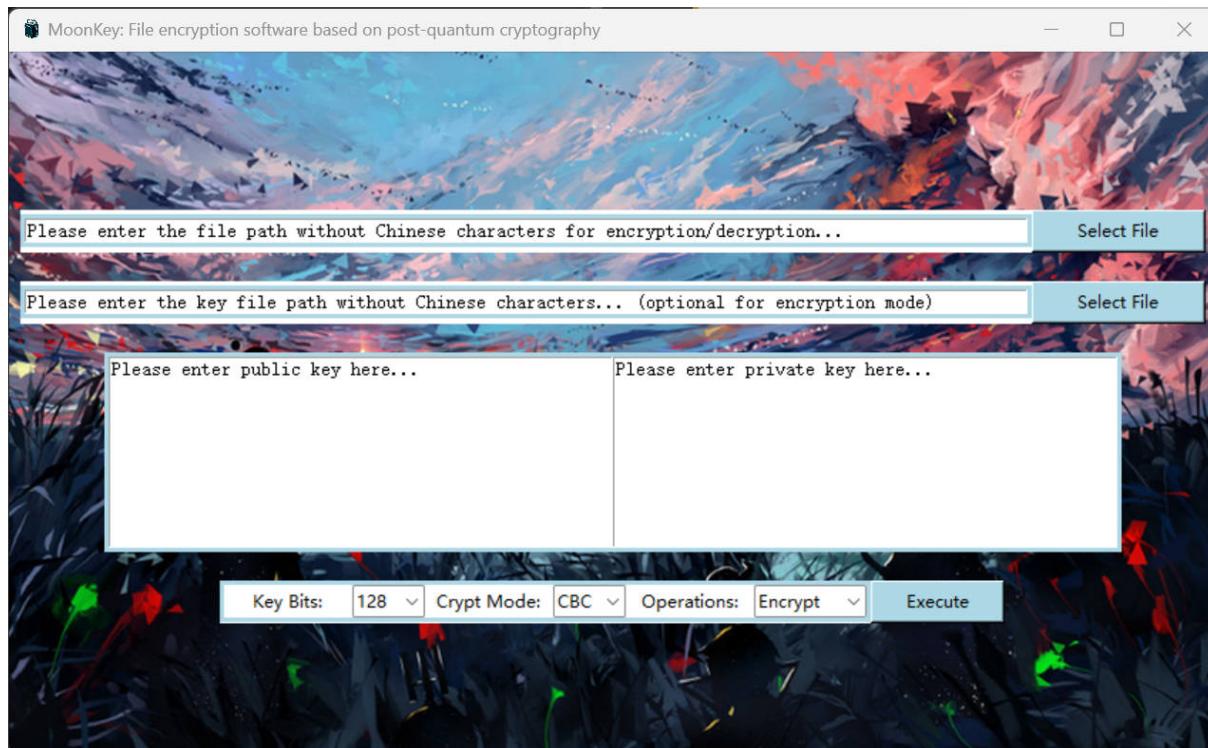


图 21: 软件UI

选择一个实例文件，这里我们随机生成了一个大小为62500KB的.bin文件，其大小和属性如下：



图 22: 待加密文件属性

选择这个文件，接下来能选择密钥大小，分组密码工作模式等，我们演示选择512-bit的密钥和CBC的工作模式，然后输入NTRU的公钥，最后点击加密按钮：

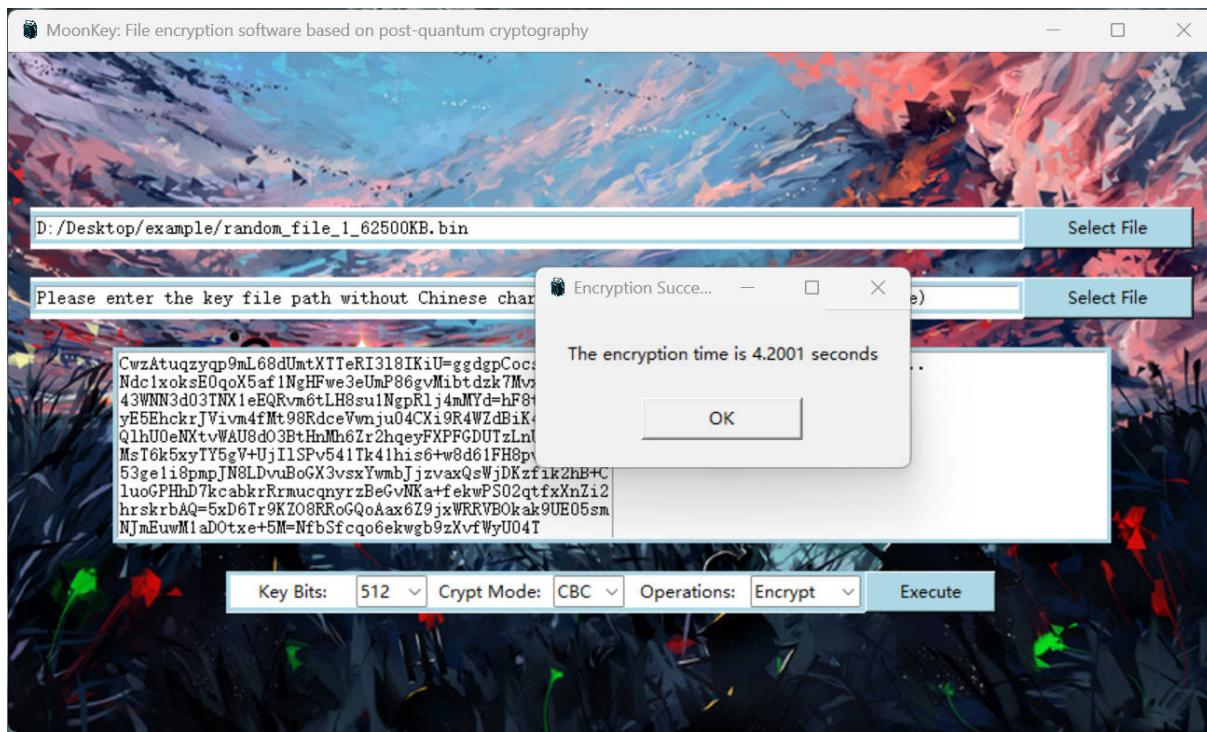


图 23: 执行加密

可以看到加密成功，加密用时4.2秒，找到文件所在目录，可以看到加密后的文件也在这里

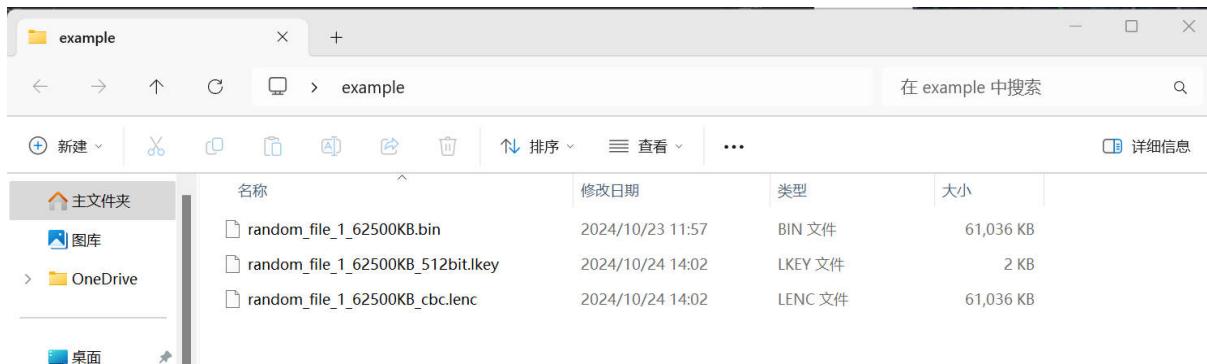


图 24: 文件目录

其中random_file_1_62500KB_cbc.lenc是密文文件，random_file_1_62500KB_512bit.lkey是密钥文件，具体属性如下：



图 25: 密文与密钥文件

这里注意密文文件比明文要多68字节，恰好是密文头部的长度。

文件解密

接下来演示解密，这里选择密文和密钥文件，输入私钥，而另外的选项中只需要把option改为解密即可，密钥长度和工作模式软件会根据密文头和密钥自动识别，如下图所示：

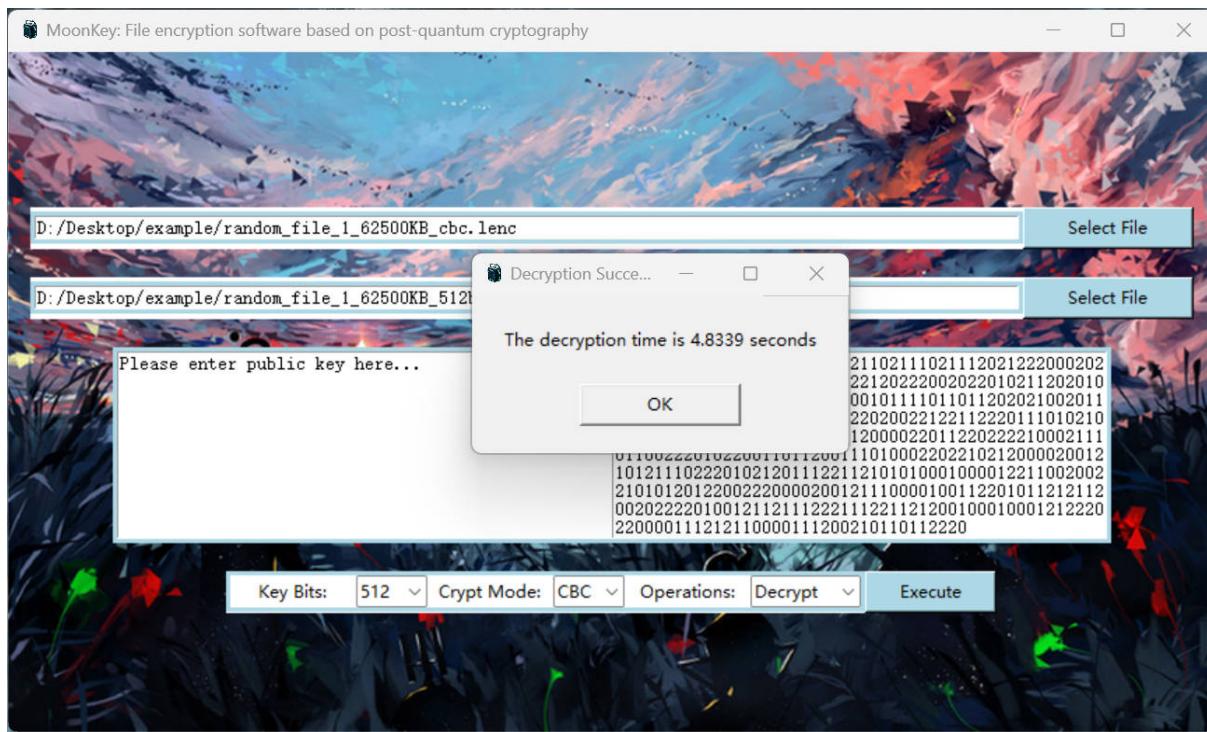


图 26: 执行解密

找到文件所在目录，可以看到解密后的文件也在这里，由于.bin是白板文件，这里解密后的文件没有被自动识别文件类型，对于一般的pdf，exe文件在解密后可以自动识别类型，这里直接展示解密后的文件属性和明文对比：

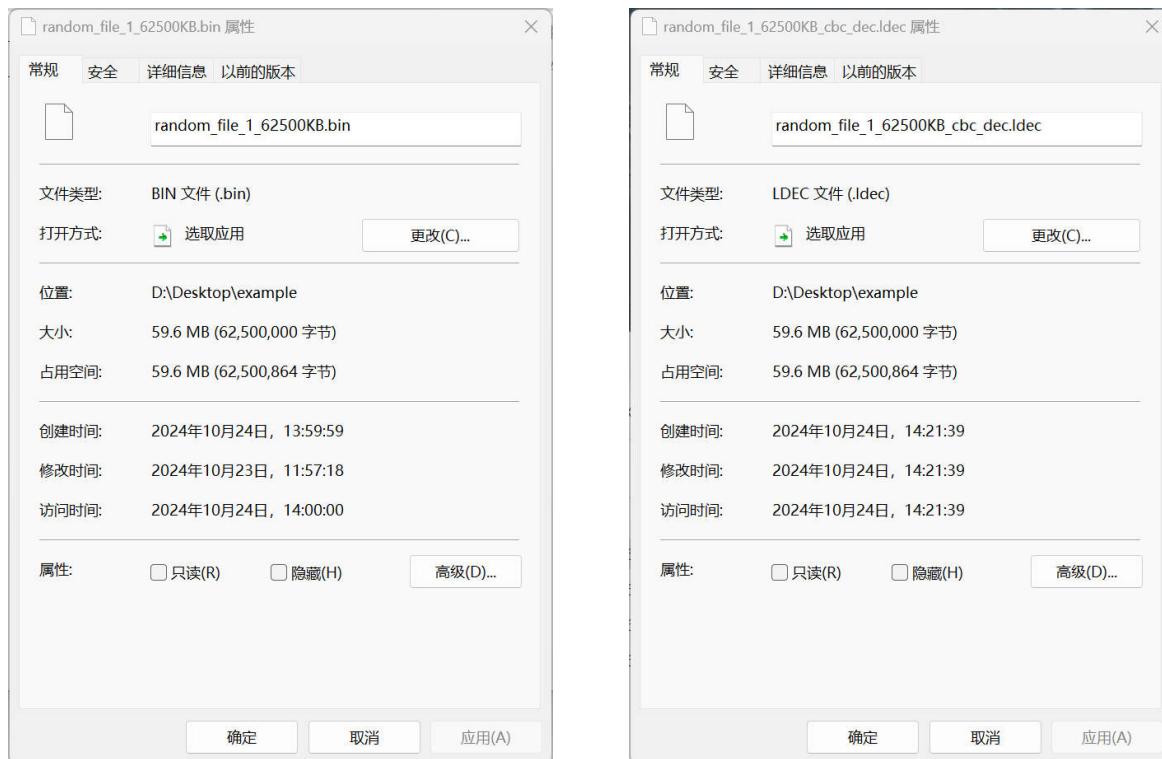


图 27: 明文与解密后的文件

文件属性上能看出二者大小一致，通过010 Editor打开查看文件比特流，发现二者相同，说明解密成功

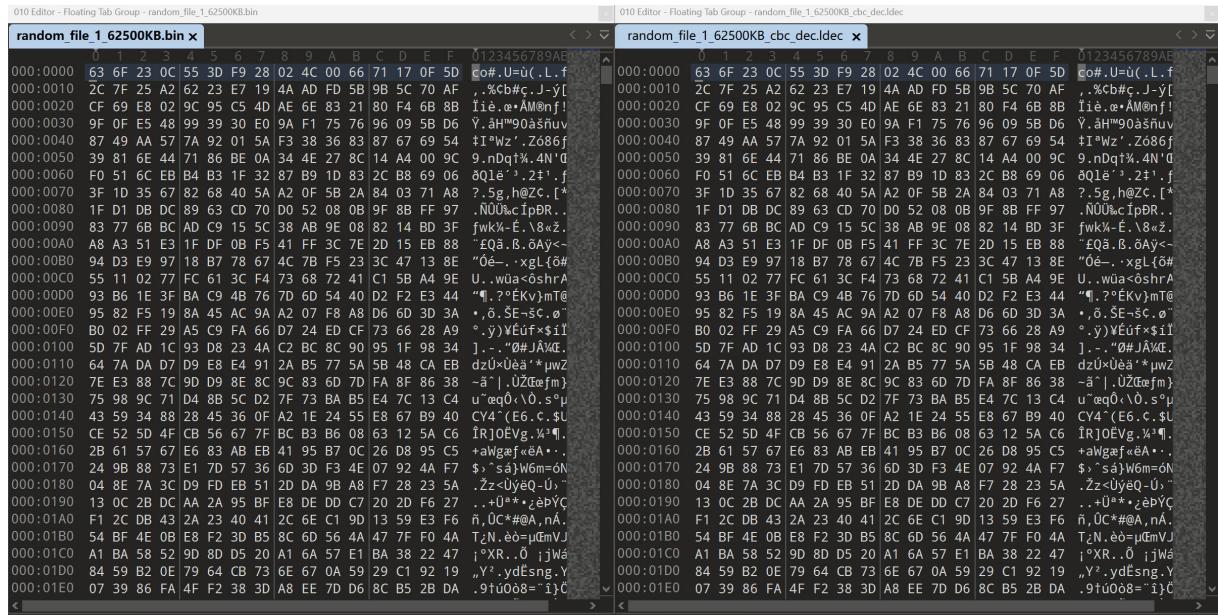


图 28: 比特流对比

3.4 性能测试

3.4.1 测试方案

接下来是对我们软件加密性能的测试，我们的测试方案如下：

- 工作模式选择CBC模式，测试变量共有：文件大小，密钥比特长度，处理器的性能
- 测试结果数据包括：加密用时，解密用时，单位均为秒
- 密钥长度我们只选择了较常用的128bit和最大密钥长度的512bit
- 在每台CPU的每个密钥长度下，我们分别选择了两种文件大小，分别是6250KB和62500KB，文件类型是.bin
- 每种大小的文件都随机生成了10个，我们都对他们测量时间最后取平均作为最终的测试结果
- 测试过程中，机器尽量关闭其他多余进程，保留系统资源，以免影响测试

3.4.2 测试数据与结果

在数据处理上，我们根据CPU性能进行分类，四台CPU的性能如下：

表 8: CPU性能

CPU测试 编号	CPU型号	主频	核数线程数
1	i5-8250U	1.6GHz	4核心8线程
2	i7-10510U	1.8GHz	4核心8线程
3	i7-1185G7	3.0GHz	4核心8线程
4	i7-11370H	3.3GHz	8核心16线程

第一台机器测试结果:

表 9: CPU1测试结果

密钥比特	文件大小	操作模式	文件1	文件2	文件3	文件4	文件5	文件6	文件7	文件8	文件9	文件10	平均
128bit	6250KB	加密	0.96450	0.87880	0.90330	0.83760	0.91170	0.89880	0.89260	0.84650	0.91960	0.91530	0.89687
		解密	0.92150	0.97720	0.95630	0.98560	0.98470	0.92700	0.98150	0.92250	0.97700	1.03610	0.96694
	62500KB	加密	8.31480	8.30830	8.37600	8.44540	8.34270	8.40720	8.31180	8.28380	8.40740	8.38870	8.35861
		解密	9.28140	9.20450	9.21010	9.25690	9.13060	9.32360	9.10600	9.23730	9.21750	9.17400	9.21419
512bit	6250KB	加密	0.90850	0.96030	0.84040	0.84830	0.87900	0.86240	0.89160	0.93300	0.92610	0.88640	0.89360
		解密	0.92310	0.91320	1.00950	0.91250	0.97340	0.94380	0.91900	0.92410	0.98800	0.92620	0.94328
	62500KB	加密	8.25070	8.25560	8.24030	8.25560	8.24000	8.25310	8.30800	8.26210	8.30180	8.27040	8.26376
		解密	9.11670	9.12920	9.16270	9.10980	9.14460	9.15990	9.16070	9.14600	9.14750	9.09650	9.13736

第二台机器测试结果:

表 10: CPU2测试结果

密钥比特	文件大小	操作模式	文件1	文件2	文件3	文件4	文件5	文件6	文件7	文件8	文件9	文件10	平均
128bit	6250KB	加密	0.92010	0.79700	0.84140	0.83950	0.87200	0.80560	0.84990	0.89440	0.84380	0.81540	0.84791
		解密	0.95440	0.91530	0.94640	1.01510	1.05800	0.97290	0.89600	1.08650	0.92990	1.02290	0.97974
	62500KB	加密	9.71380	9.60120	9.32640	9.20960	9.31520	9.17570	9.38980	9.41700	9.39650	9.14130	9.36865
		解密	10.46440	10.02300	10.45790	10.05070	10.18480	10.20220	10.03440	9.88890	10.18450	10.01520	10.15060
512bit	6250KB	加密	0.92150	0.90970	0.79940	0.86510	0.99820	0.87100	0.92290	0.89670	0.92010	0.90050	0.90051
		解密	1.03400	1.07570	0.95560	0.95650	1.01930	1.15050	0.99880	0.94430	1.03510	1.04140	1.02112
	62500KB	加密	9.42320	9.27250	9.15540	9.33110	9.64480	9.45290	9.52150	9.42420	9.40800	9.07740	9.37110
		解密	10.01660	10.20200	10.66030	10.22520	10.75670	10.07250	10.13110	10.01990	10.22110	10.07050	10.23759

第三台机器测试结果:

表 11: CPU3测试结果

密钥比特	文件大小	操作模式	文件1	文件2	文件3	文件4	文件5	文件6	文件7	文件8	文件9	文件10	平均
128bit	6250KB	加密	0.58190	0.59990	0.60150	0.58370	0.58820	0.60470	0.58670	0.57580	0.58740	0.58460	0.58944
		解密	0.67100	0.67150	0.68550	0.71840	0.68910	0.66660	0.72610	0.67500	0.66850	0.68170	0.68534
	62500KB	加密	6.02230	5.99130	5.97100	5.97180	5.93580	5.99460	5.98340	6.07480	6.14750	6.33510	6.04276
		解密	7.96170	7.18310	6.88280	7.09290	7.37410	7.29320	7.15920	7.55160	7.49090	7.42410	7.34136
512bit	6250KB	加密	0.60200	0.63490	0.59480	0.60310	0.58030	0.59890	0.58400	0.57770	0.60110	0.60240	0.59792
		解密	0.67920	0.67840	0.68610	0.67390	0.69090	0.67870	0.66540	0.67840	0.66550	0.67730	0.67738
	62500KB	加密	5.94030	6.00710	6.05490	6.04250	6.48130	6.50270	6.04140	6.01480	6.54170	5.91970	6.15464
		解密	6.69250	6.71780	7.26370	7.27350	7.48910	7.60100	7.33020	7.70660	7.40020	6.87000	7.23446

第四台机器测试结果:

表 12: CPU4测试结果

密钥比特	文件大小	操作模式	文件1	文件2	文件3	文件4	文件5	文件6	文件7	文件8	文件9	文件10	平均
128bit	6250KB	加密	0.515	0.521	0.523	0.521	0.518	0.522	0.523	0.533	0.520	0.526	0.5222
		解密	0.582	0.577	0.578	0.583	0.579	0.571	0.582	0.5853	0.580	0.575	0.57923
	62500KB	加密	5.157	5.171	5.119	5.116	5.169	5.124	5.191	5.196	5.170	5.182	5.1595
		解密	5.711	5.841	5.725	5.807	5.751	5.717	5.809	5.752	5.712	5.755	5.758
512bit	6250KB	加密	0.515	0.517	0.512	0.512	0.524	0.5163	0.516	0.521	0.523	0.526	0.51823
		解密	0.585	0.583	0.579	0.572	0.581	0.576	0.585	0.576	0.579	0.581	0.5797
	62500KB	加密	5.157	5.177	5.231	5.235	5.133	5.188	5.381	5.2182	5.189	5.155	5.20642
		解密	5.795	5.743	5.776	5.770	5.791	5.794	5.806	5.856	5.7521	5.805	5.78881

从测试结果可以得出如下几条结论

- 加解密的性能与密钥比特无关，这点也能从LBC算法中密钥比特分配方法上证实
- 软件在高CPU主频上加解密速度更快，加解密速度直接与硬件配置相关
- 解密文件用时相较加密文件用时会更长一些
- 文件大小也与软件运行速率有关，文件大10倍后软件运行慢接近十倍，说明呈现一定线性相关性
- CPU2的测试机器使用年限长，老化严重，硬件性能有所下降，因此在性能表现上不如CPU1

最后我们以CPU1用512-bit密钥加密62500KB的文件为例，计算一下每秒我们软件可以加密的分组数，根据每个分组大小为512-bit，即64字节，可以计算出一个62500KB包含多少个分组：

$$\frac{(62500 * 2^{10})B}{64B} = 62500 * 2^4 = 10^6$$

加密一个这样的文件要8.26376s，因此可计算得出平均一秒加密分组数：

$$\frac{10^6}{8.26376} \approx 1.21 * 10^5$$

可见我们的软件每秒在性能表现最低的PC上每秒加密的分组数要高于 10^5 ，具备非常高的效率

4 作品总结

随着量子计算技术的发展，传统密码算法面临潜在威胁，我们参考了ElDeen[22]，Abe[23]，Kurosawa[24]等人在混合加密的研究并做了改进，基于SM3和SM4设计出对称

密码的Feistel结构，使其能支持512-bit分组的加密运算，后又实现两种工作模式使我们的算法能加密大文件。在对密钥的处理上，我们引入NTRU抗量子公钥加密算法，通过数字信封的形式保障密钥传递的安全性，最后设计出软件UI和整体框架进行前后端连接，实现了具备实用性和安全性的大文件加密软件。

实现组合式大分组分组密码算法 该加密方案组合了SM3和SM4，密钥达到512bit。其中，SM3是一种安全的哈希算法，其设计具有良好的抗碰撞性和抗篡改性。尽管量子计算机对哈希函数的攻击能力增强，但SM3在现有条件下依然提供了相对较高的安全性。SM4是一种分组对称加密算法，具有较高的效率和安全性。量子计算机可能对对称加密构成威胁的主要方式是通过Grover算法，这种算法可以将密钥搜索的复杂度从 2^n 降低到 $2^{(n/2)}$ 。然而，通过使用512位的密钥长度，SM4显著提高了抵御量子攻击的能力，即便在量子计算机环境下，破解这样长度的密钥仍然需要极大的计算资源和时间。结合SM3和SM4的方案，能够在某种程度上增强抗量子攻击的能力。特别是对于提高密钥长度和复杂度的设计，保障了加密算法的安全性。

设计数字信封 本作品考虑到随机生成的对称密码密钥若被用户以不可靠信道传播存在被窃取的风险，引入了数字信封的设计，通过使用开源抗量子算法NTRU对用户的密钥进行公钥加密，NTRU是基于格的公钥加密，在选定好的参数下能达到十分高效的加解密，在加密高效的同时确保密钥在传播过程中的保密性，之后解密用户收到加密的密钥后，用私钥解密得到对称密码密钥，之后用该密钥对密文解密得到明文文件。数字信封的使用实现了以较小的性能消耗换取密钥传播的安全性。

软件开发 为了更好的展示方案效果，扩展其应用范围，本作品后端加密算法均采用高效的C++编程实现，基于python的GUI库开发了用户友好文件加密软件。该软件页面精美简洁，操作简单，提供了很多可选项，方便用户直接使用。

参考文献

- [1] Bernstein, D., Lange, T. (2017). Post-quantum cryptography. *Nature*, 549, 188–194.
- [2] Song, F. (2014). A Note on Quantum Security for Post-Quantum Cryptography. In: Mosca, M. (eds) *Post-Quantum Cryptography*. PQCrypto 2014. Lecture Notes in Computer Science, vol 8772. Springer, Cham.
- [3] Hamid Nejatollahi, Nikil Dutt, Sandip Ray, Francesco Regazzoni, Indranil Banerjee, Rosario Cammarota. (2018). Post-Quantum Lattice-Based Cryptography Implementations: A Survey. *ACM Computing Surveys (CSUR)*, 51(6).

- [4] Guan, C., Ren, K., Zhang, F., Kerschbaum, F., Yu, J. (2015). Symmetric-Key Based Proofs of Retrievability Supporting Public Verification. In: Pernul, G., Ryan, P. Y A., Weippl, E. (eds) *Computer Security – ESORICS 2015*. ESORICS 2015. Lecture Notes in Computer Science, vol 9326. Springer, Cham.
- [5] Coppersmith, D., Shamir, A. (1997). Lattice Attacks on NTRU. In: Fumy, W. (eds) *Advances in Cryptology — EUROCRYPT ’97*. Lecture Notes in Computer Science, vol 1233. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-69053-0_5
- [6] Sendrier, N. (2017). Code based cryptography: State of art and perspectives. *IEEE Security & Privacy*, 15(4), 44-50.
- [7] McEliece, R. J. (1978). A public-key cryptosystem based on algebraic coding theory. Jet Propulsion Laboratory, Pasadena. DSN Progress Reports 4244, 114–116.
- [8] Hoffstein, J., Lieman, D., Pipher, J., Silverman, J. (2001). NTRU: A public key cryptosystem. In Proc. EUROCRYPT, Innsbruck, Austria, pp. 211–225.
- [9] Cabarcas, D., Weiden, P., Buchmann, J. (2014). On the Efficiency of Provably Secure NTRU. In: Mosca, M. (eds) *Post-Quantum Cryptography*. PQCrypto 2014. Lecture Notes in Computer Science, vol 8772.
- [10] Bailey, D.V., Coffin, D., Elbirt, A., Silverman, J.H., Woodbury, A.D. (2001). NTRU in Constrained Devices. In: Koç, Ç.K., Naccache, D., Paar, C. (eds) *Cryptographic Hardware and Embedded Systems — CHES 2001*. Lecture Notes in Computer Science, vol 2162. Springer, Berlin, Heidelberg.
- [11] Ding, J., Petzoldt, A., Schmidt, D.S. (2020). *Multivariate Public Key Cryptosystems*, 2nd edn. Springer, New York.
- [12] Chinese National Standard, *SM3 Cryptographic Hash Algorithm*, GB/T 32905-2016, State Encryption Management Bureau, 2016.
- [13] Chinese National Standard, *SM4 Block Cipher Algorithm*, GB/T 32907-2016, State Encryption Management Bureau, 2016.
- [14] Wang, Z., Dong, H., Chi, Y., Zhang, J., Yang, T., Liu, Q. (2021). Research and Implementation of Hybrid Encryption System Based on SM2 and SM4 Algorithm. In:

- Liu, Q., Liu, X., Li, L., Zhou, H., Zhao, HH. (eds) Proceedings of the 9th International Conference on Computer Engineering and Networks. Advances in Intelligent Systems and Computing, vol 1143. Springer, Singapore. https://doi.org/10.1007/978-981-15-3753-0_68
- [15] Shor, P. W. (1999). Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Review*, 41(2), 303-332.
- [16] Ables, K., Ryan, M. D. (2010). Escrowed Data and the Digital Envelope. In: Acquisti, A., Smith, S. W., Sadeghi, A. R. (eds) Trust and Trustworthy Computing. Trust 2010. Lecture Notes in Computer Science, vol 6101. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-13869-0_16
- [17] Kaliski, B. (1998). PKCS# 7: Cryptographic message syntax version 1.5[R].
- [18] 庞林源, 赵亮. (2024). 国外后量子密码技术发展动向及启示[J]. 网络安全技术与应用, (06), 40-42.
- [19] 冯艺萌, 刘昂. (2024). 迈向量子安全: 后量子密码迁移研究与思考[J]. 计算机技术与发展, 34(05), 103-108. DOI:10.20165/j.cnki.ISSN1673-629X.2024.0047.
- [20] 袁晓云, 安银实. (2024). 运营商亟须提升应对量子计算攻击的能力国外后量子密码应用情况分析及对我国运营商的启示建议[J]. 通信企业管理, (07), 14-16.
- [21] 郑鉴学, 张道法, 徐松艳, 等. (2024). 基于NTRU密钥协商协议设计[J]. 信息安全研究, 10(01), 12-19.
- [22] Ali E. Taki El Deen, "Design and Implementation of Hybrid Encryption Algorithm", International Journal of Scientific & Engineering Research, vol. 4, no. 12, pp. 669-673, December 2013.
- [23] Abe, M., Gennaro, R. & Kurosawa, K. "Tag-KEM/DEM: A New Framework for Hybrid Encryption." *J Cryptol* 21, 97–130 (2008). <https://doi.org/10.1007/s00145-007-9010-x>
- [24] Kurosawa, K., Desmedt, Y. (2004). "A New Paradigm of Hybrid Encryption Scheme." In: Franklin, M. (eds) *Advances in Cryptology – CRYPTO 2004*. CRYPTO 2004. Lecture Notes in Computer Science, vol 3152. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-28628-8_26