实验报告（RSA）

【实验目的】

1、 掌握 RSA 算法原理及实现；
2、 了解常见的 RSA 攻击方法。

【实验环境】

Python 3.9.7 64-bit

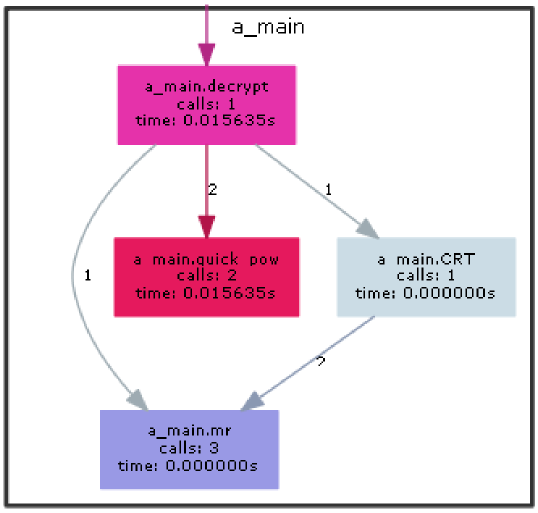
【实验内容】

一、RSA 算法

1、算法流程

- 加密：$c = m^e \bmod p \cdot q$；
- 解密：$m = c^d \bmod p \cdot q$，其中 $d = e^{-1} \bmod (p-1)(q-1)$。

函数调用关系图如下：



函数名	函数功能
quick_pow()	快速模幂
CRT()	中国剩余定理
mr()	求模逆

2、测试样例及结果截图

```
9084765279459973722703105122578434463570460548446011782795531096631608711386758039869846624407809636935725454210198201714484752683287324692998717028735972189703006864635333180338384765812830167319173319
6580950392194561201082122223798322385484142722546227274126019868335307255095676562482247070488705415319999541622125183113185633569436801807262284795751129437784576275843041325259889627849392531691924872920409522402926597627191488120636244538724270022998166981037583694182889745072153997951709132903469153746154058782302049361904975953920415583179235627460458533897822450783730009468017406778947855134635
8267188997198709383102216295528861867050429668709122381362481797742114635297750
1
5616976246200305158039112558279347117127190323140566029817119222816875304862873433160440188246954626180457052713652373546503116990250870468693721436449417595407928058472984986636283252068375379203054671193294002273552318509530075387982542503310814623494784422486899345631204592432795069494302263847732614570838788115433044853549920244063677931167782491853844961979279728902617687381203583457549865267530

Process finished with exit code 0
```

```
3145146953629048607738364782752285594106365442120564670006583827002441998129206649536184601915044443620
 834616226503929392043067133733622624903785033696825742023625905351404449254518300949588957517542057З
271510317494885375407951621418157800057887022565155717188045596860779984033972368709891024667850677188
 8275180254757552331445027383595590396150885596947416571314897563955799952947290635642290806606383147
835307195534669747385428658241880542971548837167443854348790393026347857446401853669046322331046283620
 9978665917822358744770565680255133448695933964392087478519876793678011898365315127216008242280076273
610005905275766179257772352453126536248349042313487737529081650312252270770413132137809871870224160611
 58991824262470164711838317972173679067619580422685953060503453497213650918902889131660088284579816788
 96914533049036894858915683951643003086886242119353076345361949019179184935042225352581657385633372397
 929542454785761578018399699067741498701021633164084571749813075321648182679566573019789792234366293
0
15207835352836962566843817308506642859032535621100422165482455033137859330220880

Process finished with exit code 0
```

3、讨论与思考

- 在解密过程使用中国剩余定理加快解密算法。
- 在**本地**通过 Miller-Rabin 素性检测生成大素数 p,q，随机寻找与 $\varphi(n)$ 互素的 e；

生成大素数规则：

- 先在 $2^{1024} \sim 2^{1025}$ 范围内选取一个数 p，如果找了 100 次的 a 都不是 p 的“Miller-Rabin witness”，则认定 p 为素数；

安全性分析：

- 由 Miller-Rabin 素性检验我们知道：若 n 是一个奇合数。那么取值在 1 到 $n-1$ 之间的 a 至少有 75%的概率是的 Miller-Rabin witness。
- 这说明如果 n 是合数，那么每次尝试 a 的值都至少有 75%的机会得到 witness，由于在 100 次试验中都没有发现 witness，因此很有理由断定 n 为合数的概率最多为 $(25\%)^{100} \approx 10^{-60} \rightarrow 0$，在这种情况下，我们有充分的把握相信 n 为奇素数。

本地生成 p,q,e 截图：

```
p:
 25208654087633668454670495137054075452563908534857952670328888300348079031492
 02300149187873410972491564679281933040800677648102642695964246937715514422010
 33414737311400895841692850570317887178896590403655945474348449225671568713675
 482455992605173498205474521195631993947500779019054667519496758539359975455907
q:
 24665789140459691306223190005209772698480037295174356882135150456110441248897
 53968006210471599922934195452771495940633894381112232495191615757841208387668
 15372194503552368847026761454663266768307113644330910827545470988604125778611
 263817040682326669560180740210253823602818196783801513302076701692609832575213
e:
 45498492069391914595672663375728136326969574397523185820545473523916515043342
 07210784027110754498501799147299075951949560959771306585949440022528441931657
 81213172210781664281547627889903193849538498717717007436431639330283150076136
 60012377418639147762133343657938524598762564865879823080813820489857855892814
 54197278528112213817940459003615434230057902977082696644762598278875350500992
 65496709349689290758293028940698753869571966787344037450070083086847626899713
 00427246867290785016425259228184847492759688145902447309806083726930744515872
 708516030054637783885060315061797783356709317118332096190737857076891358200005
```

二、小指数广播攻击

1、算法流程

- 加密指数e非常小，一份明文使用不同的模数N，相同的加密指数e进行多次加密；
- 可以拿到每一份加密后的密文c_i和对应的模数N_i，加密指数e：

$$
\begin{gathered}
m^e \equiv c_1 \bmod N_1 \\
m^e \equiv c_2 \bmod N_2 \\
\cdots \\
m^e \equiv c_n \bmod N_n
\end{gathered}
$$

- 通过 CRT 求出m^e后开根即可。

2、测试样例及结果截图（结果冗杂，仅展示 OJ 截图）

评测编号 ↓	提交时间	提交状态	代码语言	最大运行时间	最大运行内存	详细信息
20692	2022-05-11 19:31:18	Accepted	Python	44ms	10160KB	

3、讨论与思考

- 听说 `pow()`精度不够，直接用的 `gmpy2.iroot()`开根。

三、共模攻击

1、算法流程

- 已知：

$$\begin{cases} c_1 \equiv m^{e_1} \bmod N \\ c_2 \equiv m^{e_2} \bmod N \end{cases}；\text{且} \gcd(e_1, e_2) = 1$$

- $\exists\, s_1, s_2$，**s.t.** $e_1 s_1 + e_2 s_2 = 1$，故：

$$\begin{aligned} \Rightarrow c_1^{s_1} \cdot c_2^{s_2} &\equiv \left(m^{e_1} \bmod N\right)^{s_1} \cdot \left(m^{e_2} \bmod N\right)^{s_2} \bmod N \\ &\equiv \left(m^{e_1}\right)^{s_1} \cdot \left(m^{e_2}\right)^{s_2} \bmod N \\ &\equiv m^{e_1 s_1 + e_2 s_2} \bmod N \\ &\equiv m \bmod N \end{aligned}$$

- 由扩展 Euclid 得到s_1, s_2，用快速模幂计算$c_1^{s_1} \cdot c_2^{s_2} \bmod N$即可。

2、测试样例及结果截图

```
346270541728252349966689606661927676577210124792353
595614202586059875656387022106783821468266573506693
154955311000579732437188648319131994792342638009003843285187270576012791286122494146996214447606784
80754113864809618536150917127548539061044955830752723108781043149087830769088238710057791316026968
156901705893430563139409154971022074895105669127211169712652765533727765305030716855665951223742613
55191645467685740589271889229103998923005943759459


Process finished with exit code 0
```

```
959559461963512608443126267894722577435595927353245
453561406060629441513517787631500095541985432999623
93348788151179115999777658841992375566337169603613545826775828941873088126151158391355982090949056
195535073200740134718658864013777573590333481680107472823411070838887551713826888289490054914765135
284692435486579387446059868209966611187130207151707238183764764200229735175022665877552838862750631
52111229549912194077437903675618092345052472245295


Process finished with exit code 0
```

3、讨论与思考

- 既然可以用 `gmpy2` 的话，那就直接用 `gmpy2.invert()`求逆元吧。

四、已知公私钥分解合数 N（选做一）

1、算法流程

- $\varphi(n)=(p-1)(q-1)=pq-(p+q)+1=N-(p+q)+1$；
- 由于 p, q 非常大，$pq \gg p+q$，故 $\varphi(n) \approx N$，$ed=k\varphi(n)+1 \approx kN$；
- 先计算 $ed \div N$ 的整数部分 k'，由于 ed 略小于 kN，我们只需要向上继续搜寻 k' 直至 $k' | ed-1$，此时 $\varphi(n)=(ed-1) \div k'$；
- 利用韦达定理，构造方程 $x^2-(p+q)\cdot x+p\cdot q=0$，其中 $p+q=N-\varphi(n)+1$，$p\cdot q=N$，这个方程的两个根即为 p, q。

2、测试样例及结果截图

```
1126574657898314166102104310359399413389749805938 75
39151878562289741205218468898235793304091908305768346031270665658364799841492017161552516458379965 1
50270158674007816504265219983726943332831512859702386440191634296624942142793848948369849014042660 7
1257604189079087936579030891936822157724056192025 3
3997295739832052816474082400583161071193295143061 9

Process finished with exit code 0
```

```
185826930866337241052147166615123554610976232886325
12931495879482530305787660467523148956315734750707978463850500192838325984007924532959344679494505 3
14203393536187674117735035836839363364491360402821167352498605666973762513944338003185060067876825 7
871497140265063631709501163204620055613898492363 3
1629769379606655761562249974992438734894725222132 9

Process finished with exit code 0
```

```
103929963256545989672634199603298311234807347956201
179805257261817637199137932761302637577744056913636962912422565389068001366412910474892299266357657
218058279296277670030729192079898819194892991859883609186146306041747880202154970884817063901362291
887524322346518830766107729540353817437621736670 9
2456927363069386512180825455748638690847189178099 9

Process finished with exit code 0
```

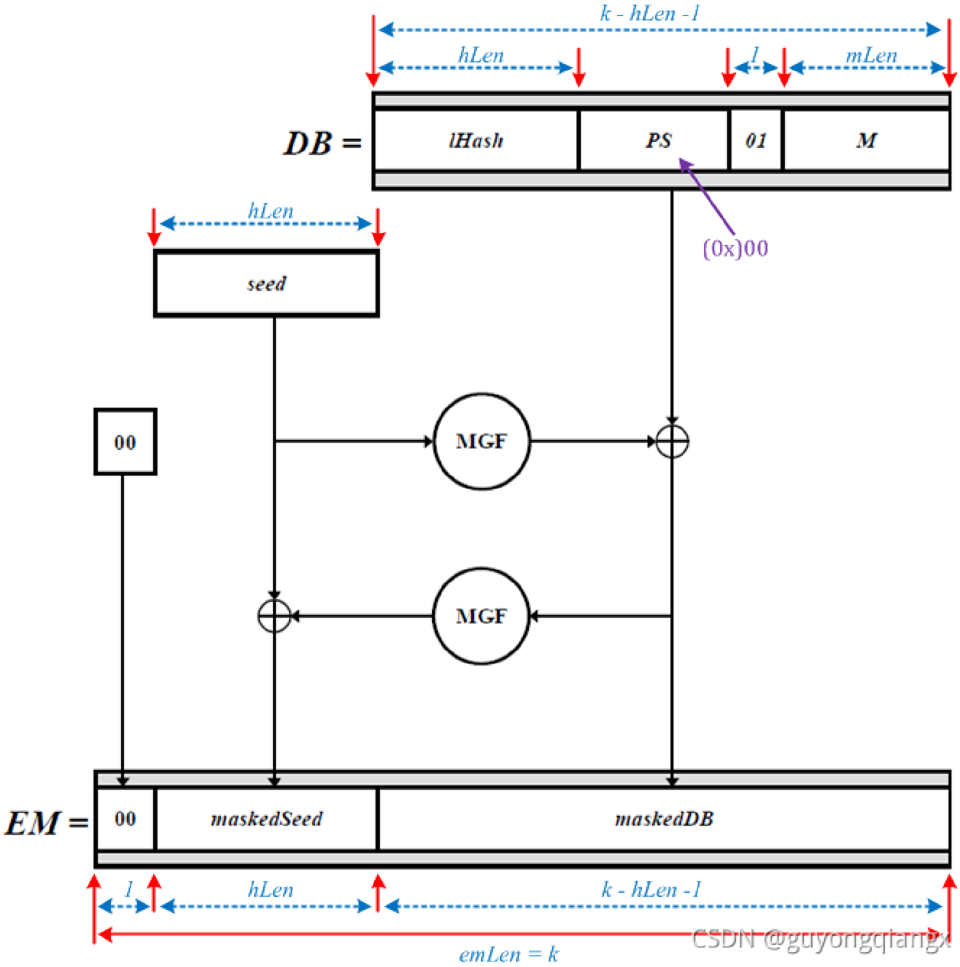
3、讨论与思考

- 我们可以通过实践证明这样搜寻 k 的快捷性，如下图所示，$ed \div N$ 的整数部分与真正的 k 是非常相近的。

```
85697978727806562921703649862823822759581931746570
85697978727806562921703649862823822759581931746584
```

五、RSA-OAEP（选做二）

1、算法流程



2、测试样例及结果截图（结果冗杂，仅展示一组样例及 OJ 截图）

```
1
128
0x10001
0xc168fb2417c03f369904f83060870eb7ee92e609b6087d8345d927579c2495618fed5c08ab41673bc6d15979
 71c1652ffced577b548fa1b889deb55a7c6907af9a9824b3001dcc9a51a449919844bf2dc57d1ae74d76b098
 70bd1c1d7bacbb762cc24d8d9c23995a3e08ee977197e1c90420ec01c3d4c5703667e91d678e2c13
0xab541d54c0f9d92c443e1835244e237f985cee56c6aefccd8c79764b1ded61d927b48d8fdf7ce8f694587988
 b88f9d0f37c11a833b3cc7b84ba9802fb057459347a5218e60ffb177550d85ea514ab6045f78e063f0c1e3b7
0xfd8a1f243c7f38fd
0xaf80389753872f27c31dc6516607d755e11d21f4
Err

Process finished with exit code 0
```

评测编号 ↓	提交时间	提交状态	代码语言	最大运行时间	最大运行内存	详细信息
20793	2022-05-12 00:43:41	Accepted	Python	246ms	10960KB	

3、讨论与思考

- OAEP——Optimal Asymmetric Encryption Padding（非对称加密填充）。

以下是我的一些小理解~~（这是上学期的）~~：

- 传统的 RSA 加解密是一种确定性的方案，加密指数e和解密指数d在传输信息过程中都是确定的；换句话说，传统 RSA 中的这些指数（包括$\varphi(n)$等）在确定性的角度来讲是“可逆”的，意思是说，我们可以通过已知的某些条件（如c, e, N等）来推知其他指数（如$\varphi(n)$, d等），从而逆推出明文m，这也是 RSA 常用的攻击手段。
- 而 RSA-OAEP 最大的特点就是引用了随机性（随机数r，且用完即销毁），以及“不可逆性”（单向函数）构建了一种随机预言模型，通过添加随机性元素，将传统 RSA 的确定性加密方案转变为了概率加密方案；通过无法反转的单向函数使得攻击者无法恢复明文的任何部分，从而防止密文的部分解密，保持了m的“全有/全无性”。

六、维纳攻击（选做三）

1、算法流程

简要证明：

- 由于$ed-1=k\varphi(n)$，两边同时除以$d\varphi(n)$，得：

$$\frac{e}{\varphi(n)}-\frac{k}{d}=\frac{1}{d\varphi(n)}$$

- 由于$\varphi(n)\approx N$，故$d\varphi(n)$也是一个大数，故$\frac{e}{N}$略大于$\frac{k}{d}$；
- 在$d<\frac{1}{3}n^{\frac{1}{4}}$且$q<p<2q$的条件下，$\frac{e}{N}$的连分式展开能够精确的覆盖$\frac{k}{d}$。

定理：

假定$\gcd(a,b)=\gcd(c,d)=1$且

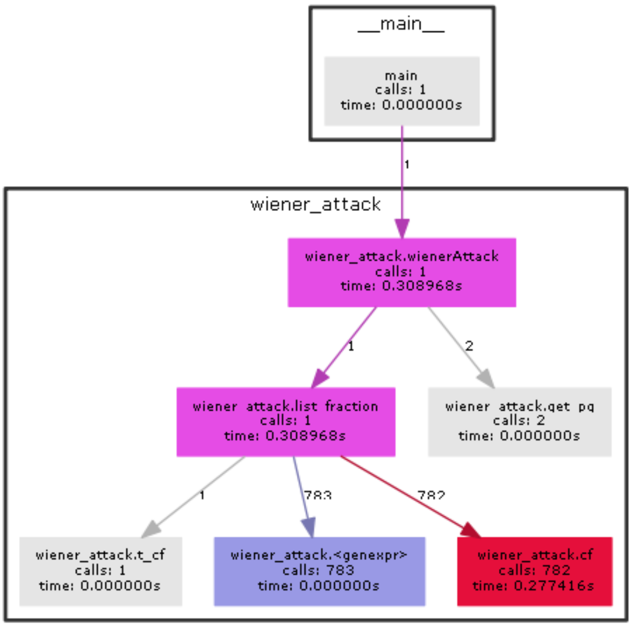
$$\left|\frac{a}{b}-\frac{c}{d}\right|<\frac{1}{2d^2}$$

那么c/d是a/b连分数展开的收敛子。

算法原理：

- 先列出e/N的每一个渐进分数；
- 判断$k|ed-1$，成立则$\varphi(n)=(ed-1)\div k$；
- 韦达定理用N, $\varphi(n)$求p, q（同“四”）。

函数调用关系图：



函数名及对应功能：

函数名	函数功能
wienerAttack()	维纳攻击主函数
t_cf()	将分数转为连分数的形式
cf()	得到渐进分数的分母和分子
list_fraction()	列出每个渐进分数
get_pq()	通过韦达定理求p, q

伪代码：

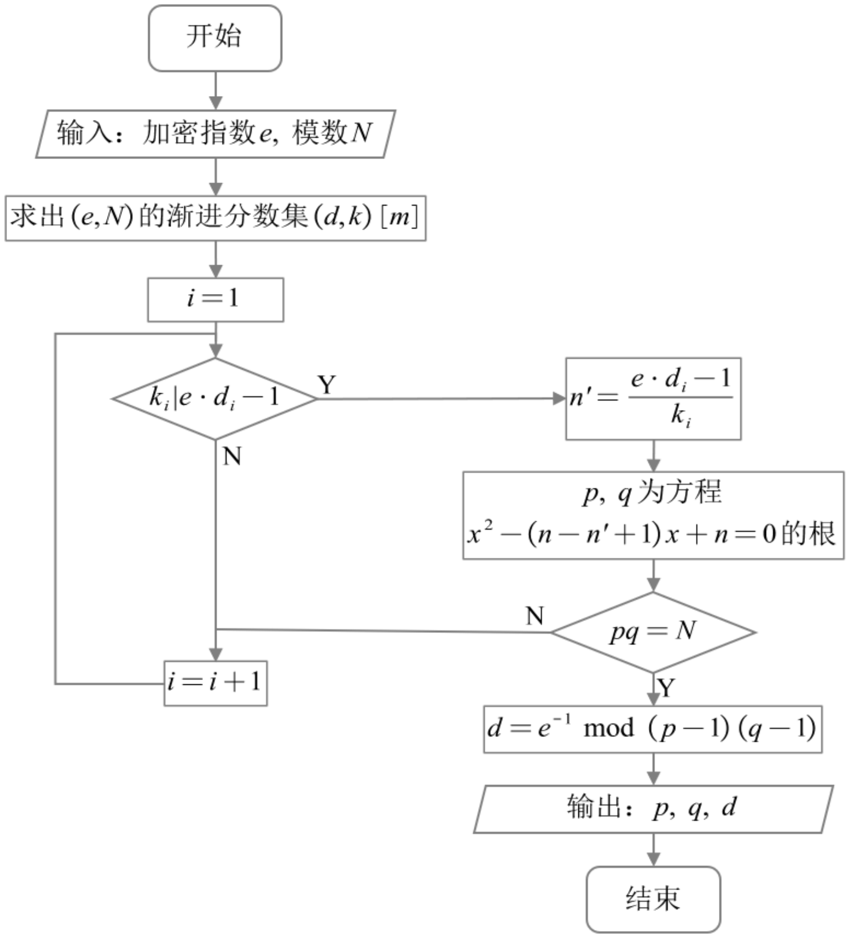
Algorithm 1: Wiener Attack

Input: Encryption index e, Modulus N

Output: p, q, Decryption index d

function "wienerAttack(e, N)"
    $(q_1, \cdots, q_m; r_m) \leftarrow$ Euclidean Algorithm(N, e) // 欧式算法求连分数的参数
    /∗ 初始化 ∗/
    $c_0 \leftarrow 1$
    $c_1 \leftarrow q_1$
    $d_0 \leftarrow 0$
    $d_1 \leftarrow 1$
    for $j \leftarrow 2$ **to** m **do**
        $c_j \leftarrow q_j c_{j-1} + c_{j-2}$
        $d_j \leftarrow q_j d_{j-1} + d_{j-2}$
        $n' \leftarrow (e \cdot d_j - 1) / c_j$
        /∗ 如果c_j / d_j是正确的收敛子，则$n' = \varphi(n)$ ∗/
        if n'为整数 **then**
            设p, q为方程$x^2 - (n - n' + 1)x + n = 0$的根
            if $p \times q == N$ **then**
                $d \leftarrow e^{-1} \bmod (p-1)(q-1)$
                return p, q, d
            end if
        end if
    end for
    return False
end function

流程图：



2、测试样例及结果截图（结果冗杂，仅展示 OJ 截图）

评测编号 ↓	提交时间	提交状态	代码语言	最大运行时间	最大运行内存	详细信息
20794	2022-05-12 00:43:54	Accepted	Python	100ms	10432KB	

3、讨论与思考

- 这个式子是维纳攻击的核心：

$$\frac{e}{N}-\frac{k}{d}=\frac{1}{d\varphi(n)}$$

- 等式右边的分母很大，作为整体很小，意味着等式左边的减数和被减数的差距很小很小，并且可以通过被减数的连分数求解**不断逼近**它本身的一个渐进分数，因此可能会存在某个渐进分数可以满足减数的要求；
- 当然按照求解的渐进分数的分子分母分别对应减数的分子分母，因此从头将所有的渐进分数的分子分母求解出来。

【收获与建议】

1、收获

- ~~加深了对 RSA 的理解。~~
- ~~提高了对 Python 的熟练度。~~
- ~~巩固了排版能力。~~
- 收获了劳累与繁忙。

2、建议

- 无。

【思考题】

1、 考虑 RSA 算法在实际应用中提高安全性的措施。

- 若使 RSA 安全，p 与 q 必为足够大的素数，使分析者没有办法在多项式时间内将 N 分解出来。建议选择：
  - p 和 q 大约是 100 位的十进制素数。
  - 模 N 的长度要求至少是 512 比特。

2、 RSA 算法在生成密钥时为什么要选取大素数？请简要说明。

- 为了抵抗现有的整数分解算法，对 RSA 模 N 的素因子 p 和 q 有如下要求：
  - $|p-q|$ 很大，通常 p 和 q 的长度相同；
  - $p,\ q$ 为强素数。
- 一个素数 p 是强素数需要满足以下三个条件：
  - $p-1$ 有大的素数因子，称为 r（存在 $p-1$ 分解算法，这一算法只在模 N 存在一个素数因子 p 满足 $p-1$ 平滑时有效）；
  - $p+1$ 也有大的素数因子（存在 $p+1$ 分解算法，这一算法只在模 N 存在一个素数因子 p 满足 $p+1$ 平滑时有效）；
  - $r-1$ 仍然有大的素数因子（为了保证循环攻击无效）。

3、 阐述如何利用 RSA 算法的性质进行选择密文攻击。

- 利用 RSA 算法的乘法同态性：对 $\forall x_1, x_2 \in \mathbb{Z}_n$，

$$E_K(x_1 \cdot x_2) = E_K(x_1) \cdot E_K(x_2)$$

- 选择密文攻击将希望解密的信息 C 伪装成 $r^e C$，让拥有私钥的实体解密。然后，经过解密计算就可得到它所想要的信息。

$$\begin{aligned}(r^e C)^d &\equiv r^{ed} \cdot C^d \bmod n \\ &\equiv r \cdot M \bmod n\end{aligned}$$

$$\Rightarrow M \equiv (r^e C)^d \cdot r^{-1} \bmod n$$