1 第二讲 公钥密码学：RSA

在密码学中，我们有一个重要的假设：给定一个加密系统，我们总是知道它是如何设计的。

在对称密码中，我们的核心议题是试图找到尽量“随机”的置换。从恩尼格玛机器到 AES，这个基本的思路从未改变过。

在 1960 年之前，密码主要由政府机构使用，密码本、密钥交换并不是一个很大的问题。但在六十年代之后，私人计算机出现，我们开始有了一个极其巨大的计算机网络，在这样的情况下，我们又如何才能进行密钥分发?

Steven Levy 的《Crypto》介绍了有关现代密码学出现的历史。推荐大家去看一看。

公钥密码学的思想最早由迪菲 · 赫尔曼给出。他并不是非常优秀的数学家，提出了公钥加密的思路，却不知道如何去实现。最早的公钥加密方案由 R、S、A 三人给出，一个是工程师，一个是计算机学家，一个是数学家。姚期智先生从 MIT 辞职时，Shamir 得到了他的职位，从而和 MIT 的另外两名教授（R 是他的老朋友）见面。有一天 Shamir 在图书馆中发现了迪菲的论文，告诉 R 我们应该试着做成这件事情——他们又找到 Adleman 合作。

Shamir 在论文的最后他留下了一句话：如果读者期望得到更多信息，可以写邮件给我（当时还没有电子邮件）。他把论文发表了之后就出去度假了——当他回到大学，发现自己的办公室里已经堆满了数千封邮件。他此时终于意识到自己做出了多么伟大的发明。

下面我们来系统地介绍公钥密码学。首先介绍 RSA。

假设小明希望使用 RSA 系统来进行加密通讯。他首先需要选取两个很大的质数 p, q（长度至少为 1024 比特），然后取 $n = pq$。接下来，他还需要找到一个 e，满足：

$$\gcd(e, \phi(n)) = \gcd(e, (p-1)(q-1))) = 1.$$

小明将公开他的公钥：(n, e)。他还将藏好自己的私钥:d，满足 $de = 1 \mod \phi(n)$。

在 RSA 加密方案中，明文空间为 Z_n；加密函数为：

$$m \to m^e \mod n = c \in Z_n.$$

这个加密过程是可以被持有公钥的任何人完成的——只要知道整数 n, e 就可以。接下来，得到密文 $c = m^e$ 的小明只需要计算：

$$c^d = m \mod n$$

即可得到信息 m。

我们计算一个具体的例子。选取两个质数为 $5, 11$，取 $e = 3$ 与 $\phi(n) = 40$ 互素。公钥对为 $(55, 3)$；私钥为 27。假设密文是 31，加密得到 $31^3 = 36$；解密只需计算 36^{27}。我们并不需要计算 27 次乘法——我们只需要使用快速幂算法。

$$36^{27} = (36)^{2^4} \times (36)^{2^3} \times 36^2 \times 36.$$

需要特别注意的是，当 $\gcd(m, n) > 1$ 时，解密函数并不能得到 m，但这件事发生的概率可以被认为是 0.（使用 1024 位的密钥，概率约为 10^{-300}，这可以被认为是不可能事件。）

下面我们讨论 RSA 系统的安全性。十分显然的是，如果知道 $\phi(n) = (p-1)(q-1)$，则我们可以计算出 p, q（反过来也一样）。如果我们能对大整数做素因子分解，我们可以知道 p, q，也就破解了这个加密方案。反过来，如果我们能有效地计算出 d，我们也能找到对 n 的分解（这一点不那么容易看出）。但是，值得强调的是，我们并不一定需要找到私钥才能破解密文（我们尚未证明或者证伪这件事）。因此，破解 RSA 并不等价于大整数分解问题。

最后，我们讨论如何找出两个大素数 p。根据素数定理：

$$\lim_{N\to\infty} \frac{\pi(N)}{N/\log N} = 1.$$

因此，如果我们使用长度约为 $exp(300)$ 的素数，在大奇数中随便取一个，总有约 1/150 的概率找到素数。我们只需要随便取一些数，然后检验它们是否为素数——如果是，我们就很高兴地采用它们。同样地，尽管我们有多项式时间内判定任何整数是否为素数的方式，我们并不会采用它（耗时太长，不经济）。我们可以采用费马检验法：

$$y^{x-1} \neq 1 \mod x \Rightarrow x \in \mathbb{Z}/\mathbb{P}.$$

我们还有成功率更高（可以计算成功率）的素性检验法（Miller-Robin test），在实践中我们一般采用这些方法。

下一个有关安全性的问题是，什么样的素数是安全的？我们可以给出一些不安全的素数的例子：如果 p, q 都是梅森素数，那么我们可以很轻松地破解这个加密方案。具体方法如下：随便选一个整数，希望它是 $Z_n^\times$ 的一个原根，对其反复进行（模 n 的）平方计算，寄希望于某一次计算之后得到 1——从而得到 $Z_n^\times$ 的阶数。这样尝试的成功率很高（接近 1/2）。类似地，如果我们选择一些太容易被分解（素因子太小、太少）的 $p - 1 = 2^a 3^b 5^c ...$，攻击者同样可以对随机的整数进行快速幂运算，并指望在很短的时间内尝试出一个结果。

最后，我们介绍通过平方筛法破解一般的 RSA 的方法。如果我们已经找到一组非平凡的 $x, y \in Z_n$，满足

$$x^2 = y^2 \mod n.$$

那么我们就可以通过计算 $\gcd(x \pm y, n)$ 来得到 p, q。下面我们使用平方筛法来找到一组这样的 x, y。定义函数 f:

$$f(x) = (x - m)^2 - n,$$

其中 m 常取为 $\sqrt{n}$ 取整。我们计算一系列的 $\{f(x_i)\}$，保留那些只有较小素因子的:

$$f(x_i) = (-1)^{a_{0,i}} \times 2^{a_{1,i}} \times 3^{a_{2,i}} \times ... \times {p_k}^{a_{k,i}}$$

我们需要找到一组 $\{f(x_i)\}$，使得 Z_2 上向量组 $\{(a_{0,i}, a_{1,i}, ..., a_{k,i})\}$ 线性相关。定义矩阵

$$A = [a_{j,i}]_{i,j=0,1,...,n}$$

，由于其行向量线性相关，必可以找到 Z_2 上的线性方程组 $A^T v = 0$ 的非零解。这组非零解给出了:

$$Y^2 := \prod_{i=1}^{n} r_i f(x_i) = (-1)^{2d_1} \times 2^{2d_2} \times ... \times p_k^{2d_k}.$$

其中 r_i =0 或 1。则有以下等式:

$$Y^2 = \prod_{i=1}^{n} [r_i((x_i - m)^2 - n)] = (\prod_{i=1}^{n} r_i(x_i - m))^2 \mod n = X^2.$$

以上的等式不能保证得到一组非平凡的 Y, X（例如，如果运气非常糟糕，我们可能会得到一组 $\mod n$ 同余于 ± 1 的 X, Y，那么我们将得不到任何有用的结果），但从概率上来说，我们相当有可能通过以上的方法得到一组有效的 X, Y 解。

在实践中，对较大的 n（我记不清最大记录是多少了...）使用平方筛法所产生的矩阵 A 行/列数往往高达 2^{30}，存储数据和解方程都带来了不小的技术挑战。

2 第三讲 公钥密码学：Diffie-Hellman

本课程第一个 Project：自行编写一个 RSA 加密方案，要求 p, q 在 128 比特以上。可以使用 Magma/C++ 或任何语言。(5 分)

回到课程内容。我们介绍一种加速 RSA 解密的方式。对于解密过程

$$c^d = m \mod n,$$

我们希望计算能够更快。由于解密时持有私钥 p, q，根据中国剩余定理，有环同构：

$$\mathbb{Z}_{p\times q} = \mathbb{Z}_p \times \mathbb{Z}_q.$$

我们只需计算

$$c^d = x_1 \mod p;\ \ c^d = x_2 \mod q.$$

在进行这两次计算中，c, d 的位数将会降低一半（p, q 的位数一般相同或非常接近），这可以极大地降低计算时间。注意到求解 $\mod p, q$ 的同余方程时只需使用辗转相除法求逆，这比求较高的幂次要快得多。关于 RSA 的介绍就到这里。

下面我们介绍 DH 密钥交换方案。这个方案的原理非常简单，即使是高中生也能听懂。在 DH 方案应用的情形中，我们假设两位通信者，A 和 B，需要在公开的网络上通讯，也就是说，他们之间交换的任何信息都可能被攻击者截获。

首先，A 和 B 需要确定（并公开发布）二元组 (p, g)，其中 p 是大素数；$2 \le g \le p-2$，且 $g \mod p$ 的阶数充分大（理想的情况是，g 是 p 的原根）。

之后，A、B 执行如下操作：

- A 取定整数 $a \in \{0, 1, ..., p-2\}$，取 $\alpha = g^a \mod p$，并公开发布 α 的值。
- B 取定整数 $b \in \{0, 1, ..., p-2\}$，取 $\beta = g^b \mod p$，并公开发布 β 的值。
- B 接收 α 之后，计算 $\alpha^b = g^{ab} \mod p$；同理，A 接收 β 之后，计算 $\beta^a = g^{ab} \mod p$。$k = g^{ab}$ 就是 A、B 共有的密钥（例如，可以将其作为 AES 等私钥加密方案中的密钥）。

DH 的安全性由以下两个问题的困难性保证：

Definition 2.1 (CDH). *在 Diffie-Hellman 框架中，已知 (p, g, g^a, g^b)，试在多项式时间内求 g^{ab}。*

Definition 2.2 (离散对数问题，DL)**.** 给定 $p \in \mathbb{P}, n, a \in \mathbb{Z}^{\times}$；$a$ 是 p 的一个原根。已知 a, b 的值，试在（关于 p 的位数）的多项式时间内求出唯一的 $i \in Z_p$，使 $a^i = b \mod p$。即：在有限域 F_p 内对给定的原根 a 和非零的 b，试在多项式时间内求离散对数 $\log_a(b)$。

其中离散对数问题至少不比 CDH 问题简单，但我们尚未知道这两个问题是否等价，但 DL 的解显然可以解决 CDH。

既然我们现在手头有了两个不同的公钥加密方案（密钥交换方案），究竟哪个更好呢? 在实践中，RSA 具有一些 Forward Security Problem。斯诺登曾经披露过美国政府有一个巨大的数据库，在其中存放了所有互联网上的 RSA 加密通讯（以及它们的公钥）。由于在实践中，同一个用户往往一直使用相同的一组 RSA 密钥 (n, e, d)，一旦对于某个特定的 n，$n = pq$ 被破解，则这名用户的所有通讯都可以被破解。相反地，在使用 DH 密钥交换方案时，我们显然会在每次使用不同的 a, b（否则每次都得到的是相同的密钥，这毫无意义），相比于 RSA 这会具有更好的安全性。

一个对 DH 非常自然的推广是，我们并不总需要在 $\mathbb{Z}_p$ 这个群上进行运算。只要我们找到了某一个有限群，在其中做幂运算十分快速，但是求对数非常困难，我们同样可以在其上运行 DH。一个最常用的例子是椭圆曲线群。

下面我们介绍 El Gamal 公钥加密方案——这是一个基于 DH 问题，但达到与 RSA 相似功能的加密方案。

Definition 2.3 (El Gamal 加密方案)**.** *El Gamal* 加密方案由如下元素构成：

- 公钥：三元组 (p, g, A)，其中 $p \in \mathbb{P}$；g 是 p 的原根；$A = g^a \mod p$，其中 $a \in \{0, ..., p-2\}$。
- 私钥：a。
- $P = \{0, 1, ..., p-1\}$。

给定 (p, g, A)，加密如下进行：选取整数 $b \in \{1, ..., p-2\}$，计算 $B = g^b \mod p$。对明文 m 进行如下加密：$c = A^b m \mod p$。最终，传输的密文是二元组 (B, c)，因此 $|C| = 2|P|$。

解密过程是直接的：$B^a = g^{ab} \mod p$；令 $x = p - 1 - a$，有：

$$B^x c = g^{b(p-1-a)} A^b m = A^{-b} A^b m = m \mod p.$$

只需计算 x 和 $B^x c$ 即可。

El Gamal 的核心思想是，在进行加密时，事实上是使用公钥给信息加上了噪音 (noise)，而只有持有私钥的人才能去除信息上的噪音。这个加密方案和 RSA 还有一点不同:RSA 中加密是双射，而 El Gamal 中加密函数仅仅是单射。我们往往并不需要加密过程是双射——我们完全可以把明文集合映射到一个比它大得多的集合。

下面我们进入公钥密码学的一个重要主题：身份验证 (Authentication)。在进行通讯时，我们希望能够获得一种对通信人身份的确证——没有人能冒充他人进行通讯，也不能假装自己没有发送过某一条信息。

我们首先非常简短地介绍哈希函数。哈希函数是将任何信息映射到固定长度信息的函数:

$$F : \{0,1\}^* \to \{0,1\}^n,$$

其中 n 是一个定值（例如 256)。对于任意信息 X，计算 $F(X) = Y$ 非常快速，但是计算 Y 的原像 $F^{-1}(Y)$ 非常困难。我们事实上对哈希函数有更强的要求：给定 F，我们无法在多项式时间内找到任何一对 $X_2 \neq X_1$，使得 $F(X_1) = F(X_2)$。这个性质被称为 Collision Resistant。

如何设计一个哈希函数? 我们首先设计一个压缩函数，它是将固定长度的信息 “压缩” 到更短长度的函数。我们可以给出一个例子（尽管不太实用）：随机选定一组 $\mathbb{Z}_2$ 上的三次多项式:

$$[f_i(x_1, ..., x_m)], i = 1, 2, ..., n$$

对于信息 $X = x_1x_2...x_m$，压缩函数将 X 映到 $f_1(X), ..., f_n(X)$。

现在我们手上有了一个压缩函数，很容易想到如何用压缩函数构造一个哈希函数——多压缩几次就行了（如果需要，就在信息末尾加 0)。很简单，但是很有效。

另一种有效构造哈希函数的方式是借助任何一种被认为是安全的加密函数 $E_k(m)$，其中 k 是密钥，m 是明文。考虑信息 $X = (X_1, X_2)$，压缩函数可以构造如下:

$$C(X_1, X_2) = E_{X_1}(X_2) \oplus X_2;$$

$$C(X_1, X_2) = E_{X_1 \oplus X_1}(X_2) \oplus X_1;$$

这样的构造方式往往都是比较安全的。

真正使用的哈希函数往往都有非常琐碎复杂的构造，我们不多加介绍。下面我们介绍一种使用 RSA 构造签名系统的方法。

给定信息 m，我们使用一个（公开的）哈希函数 H，计算 $H(m)$；像在 RSA 中一样，我们选定大质数 p, q，公开公钥对 n, e，藏好私钥 d——这将是我们的“印章”，用来在信息上刻上“这确实由我发送”的证据。在给别人发送信息时，我们还需要发送一个签名：

$$s = H(m)^d \mod n.$$

收到信息的人会同时收到二元对 (m, s)。验证签名的方式是，计算 $H(m)$ 和 $s^e \mod n$。如果二者相同，说明发出信息的人确实具有私钥 d——签名成立了。

电子签名系统在现实中有着极其重要的应用。我们每个人都在使用的私人电脑上的系统都需要常常更新，但我们怎么能保证系统的更新包确实是由微软发出的，而非某些恶意的、希望摧毁你的电脑的冒充者？答案是，微软公司发布了一个 2048 比特长度的 RSA 公钥用作自己的 RSA 签名（这个非常大的 $n = pq$ 就存放在我们的电脑上），在发送信息时，我们的电脑会自己验证信息的签名是否能对得上，从而判断信息究竟是否真正由微软发出。

一个更具有浪漫气息的描述是，如果你能够分解这个 2048 位的大素数，那你就能轻易地攻破这个世界上所有使用 windows 系统的电脑。

电子签名系统的另一个重要应用是在密钥交换方案中。我们将在下一讲中介绍应用身份验证机制的密钥交换方案（Authenticated Key Exchange, AKE）。

3 第四讲 公钥密码学：Signature

我们首先定义所谓的 One-way function.

$$F : \mathbb{F}_q^n \to \mathbb{F}_q^m$$

对给定的 x，求 $F(x)$ 十分简单，但对给定的 y，求 $x\ s.t. F(x) = y$ 是计算上困难的。

究竟什么是“困难”？这是一个基于现实世界中计算用时的模糊定义：究竟是平均的计算用时很长，还是相当大一部分的计算用时很长，还是最糟糕的情况（最容易计算的参数选取下）也极其困难？在这门课程中，我们将不会涉及这些细节上的概念。

我们提到另一个问题：如何选取 RSA 的两个素数 p, q。我们说过希望 $p-1, q-1$ 具有较大的素因子，但总体来说完全随机地指定素数是可行的（需要额外进行一些简单的检验，例如确保 $p-1$ 不能被 10^6 以下的素数整除）。

现在我们讨论电子签名系统。首先，我们定义另一种身份验证方案。假设 A 持有私钥 x；A 向全世界宣布公钥 (g, g^x)。A 希望向全世界证明自己知道 x 是什么，但又不希望在证明的过程中泄露有关 x 究竟是什么的任何信息。从这个思路出发，我们可以构建一种与 RSA 不同的身份验证方案。

首先，A 随意地选取 r 并将 $t = g^r$ 发送给 B。B 随机选取一个数 c，并将 c 发给 A。A 再发送给 B 一个数：$r + cx$。由于 B 拥有 g^r, 他可以从 $z = r + cx$ 中计算出 g^x：

$$g^z = g^{r+cx} = g^r \times (g^x)^c. \mod p$$

（注意到我们始终在 Z_p 中进行整个过程，所以最后一步是简单的，而不是 RSA 问题）这个签名系统与 DH 方案非常相似，如果 DL 问题被解决，那么它将不再安全。

现在我们对一个签名系统进行理论上的总结。Completeness, soundness, 0-knowledge，这是我们希望一个签名系统具有的三个特质：如果知道私钥，总能正确地回答；如果不知道私钥，总不能正确地回答；在问答的过程中不会泄露关于私钥的任何信息。下面我们讨论以上定义的签名系统的 0-knowledge。

如果我们允许攻击者在两次使用不同的 c 时，A 返回了相同的 r，那么签名系统不攻自破：

$$\frac{z - z'}{c - c'} = x.$$

另一个问题是，这个签名系统的验证过程是可以“伪装”的。我们完全可以首先决定好 z, c，然后计算出对应的 r，在验证时却首先发送 g^r，然后再发送 c 和 z。在公众眼里，A' 和

B 已经完成了一次签名验证，但是 A' 或许根本不是持有私钥的 A，而是 B 自己的一个小号。

从这两个例子里，我们可以看出签名系统的安全性可能受到各种各样的挑战。

下面，我们介绍基于哈希函数的签名系统。给定信息 m，我们计算：

$$H(g^x, t, m) = c.$$

其中 g^x 是公钥三元组 (p, g, g^x) 的一部分（所有计算都在 Z_p 中进行）；t 是一个“承诺”（由我发送这条信息）；m 是发送的信息。在发送信息 m 之后，我们附上签名 $(t, z) = (g^r, r + cx)$。这个签名保证这条信息能，且仅能是由我一个人（持有私钥 x 的人）发出的。在某种程度上，这是一个“自我挑战”的机制：使用哈希函数，我给自己出了一道题：给出 $z = r + cz$ 的值，这件事只有我一个人能做到，不知道 x 的人无法计算 $z = r + cx$。

验证的过程如下：通过公钥 g^x 和信息 m、承诺 t 之后，任何人都可以计算 $c = H(g^x, t, m)$，从而验证 $g^z = g^{r+cx} = t \times (g^x)^c$ 是否成立。值得注意的是，验证者无法从这个过程中还原出 x：如果要从 $z = r + cx$ 中还原 x，无疑需要知道怎么计算 $r = \log_g t$，这又是一个离散对数问题。

现在我们可以稍微讨论一下，基于 RSA、DH 加密的公钥加密方案还可以怎么改变？最简单的回答是椭圆曲线群。要得到一种不同的公钥加密方案，我们只需要（在本质相同的有限群中）改变乘法/加法计算的定义并继续沿用 RSA/DH 或者其他方案，就能得到一种新的方案。剩下的问题是，究竟有那些群运算具有 Trapdoor 函数的性质？在此基础之上，我们希望这些群具有可调整的大小（至少要能够任意大），从而加密长度不同的信息。我们会（并不那么惊讶地）发现，已知可能具有这些性质的群是很少的，我们并没有多少选择（常用的只有 Z_p 和椭圆曲线群）。

下面讨论 AKE。类似于 DH 方案这样的密钥交换方案对中间人攻击来说是非常脆弱的，在密钥交换时选择合理的电子签名系统可以解决这个问题。SSL、TLS 就是常用的 AKE 方案。

我们还对 PKI 系统进行简单的介绍。一切签名系统都需要有一个“信任的源头”——你声称这是你的密钥，其他人又怎么确认这件事情？需要有某些机构来背书。在现实世界中，各个国家具有自己的公钥，被称为 root，是信任的源头；国家会使用这个密钥来对大型单位，例如清华大学的官方密钥进行电子签名（国家认证这是清华大学的公钥，这件事情可以由任何人核实）；清华大学再使用自己（经认证）的密钥来给教授和学生们的公钥进行签名，认证这确实是清华大学教授/学生的公钥。这种机制构成了一个信任的链条。我们有时会见到某个网页“不具有安全证书”，这实际上是说这个网站的公钥没有被更高层的机构签名认证过，因此不能保证信息传输是安全的。

一个是公钥加密方案，一个是电子签名，这两种密码系统构成了整个现代社会通讯的基础。一个最著名的例子是比特币。（关于比特币的介绍在此处不进行叙述）

1 Lecture 5 Multivariate Cryptography (I)

Firstly, we shall review the foundation of classical (asymmetric) cryptography. We have introduced, in the previous lectures, about 2 difficult problems that have been currently assumed to be unsolvable in polynomial time: Large number factorization problem, and Discrete logarithm problem. The difficulty of these two problems are the foundations of RSA and Diffie-Hellman encrytion scheme. You might want to ask a question: what will happen if these two problems are solved in the future? In short, the answer for this problem is PQC (Post-Quantum Cryptography).

This lecture will now move on to PQC. We have no time to introduce PQC in details, but I would like to mention a few names. Firstly, Richard Feynman, perhaps the most important figure in anything related to quantum and most interesting scientist, who described the idea of quantum computing in 1981, at a conference hosted by MIT' s Laboratory for Computer Science. I highly recommend the book *Surely You're Joking, Mr. Feynman!* for all of you to read.

As the idea of quantum computing appeared in 1980s, an important question for this idea is: what problems can quantum computing do better (than classical computing)? In 1980-90, nobody known the answer, and quantum computing is just an interesting idea. However, in 1994, Peter Shor from MIT suddenly published a paper, which gave an expected answer for that question.

The answer is, quantum computer can solve factorization problem as well as DL in polynomial time.

In Shor's original paper [3], he described an algorithm that works only on quantum computer, which solves factorization and DL in polynomial time. This algorithm, known as Shor's algorithm, is the earliest quantum algorithm in the history.

In 2000, Issac Chuang came to MIT from IBM. This man spent 15 million USD to build a quantum computer (prototype) with 7 "Qbit"s, to complete one simple task: factorize

$$15 = 3 \times 5,$$

using Shor's algorithm. At the same year, I got a tenure in the USA, and I felt like it is time to try something new other than algebra (representation theory, more precisely), so I dived

into the area of PQC, which aim to find replacement for encryption schemes based on DL, factorization (in Zn or elliptic curve group) in the future.

in 2006, the first PQC conference had been held and in 2008 there was the second. At this time, people have not paid much attention on PQC, but everything changed in 2015 Aug 19th, when the US government announced that they want to replace the current encrytion standard. This decision was out of nowhere, and we don't know why this happened, but anyway it made PQC a hot field. NIST(National Institute of Standards and Technolog) set the deadline on Nov 31st, 2017, and mathematicians all around the world started to submit PQC schemes to them. On July 2022, NIST announced the first group of winners:

CRYSTALS-Kyber, lattice crpto, for PKE/KEM;

CRYSTALS-Dilithium; FALCON, lattice crpto, for signature;

SPHINCS, Hash-based, for signature.

And June 2023, NIST announced another round of qualifiers, including UOV (Unbalanced Oil and Vinegar) scheme. This is a crpto scheme based on multivariate equation. I am very confident that UOV will be standardized, perhaps in 10 years, every PC will be equipped with UOV scheme. In this course, we will first cover the idea of Multivariate, and then Lattice.

(I should mention here that 3-SAT problem, a known NPC problem, is *more or less* equivalent to solving multivariate equation in $GF(2)$. This equivalence is tricky because the coefficient size before and after problem reduction might change drasticly, and it is hard to define whether one problem can be reduced to another in polynomial time.)

Now we start talk about Multivariate equation in encryption. In most cases, we consider quadratic multivariate equations. Why not higher degree? Here is an easiest example:

$$x_1x_2x_3 = 5$$

One can easily transform it to a quadratic equation with 1 more variable y:

$$x_1x_2 - y = 0; x_3y = 5.$$

It is obvious that all multivariate equations with degree higher than 2 can be reduced to quadratic equation by adding more variables, so it is meaningless to use higher degree.

1.1 Tame Transformation and Encryption

Since it is well-known that solving multivariate equation is computationally difficult (or more officially, NPC-hard), many cryptographers, including Diffie and Tsutomu Matsumoto had tried to design crpto schemes based on it. An idea is to use a special type of transformation, called Tame transformation:

$$F(x_1, x_2, ..., x_n) = \begin{pmatrix} x_1 \\ x_2 + f_2(x_1) \\ x_3 + f_3(x_1, x_2) \\ x_n + f_n(x_1, ..., x_{n-1}) \end{pmatrix}$$

where f_i are quadratic polynomials of $(x_1, ..., x_i)$.

Tame transformation is a non-linear transformation of vector x to y which is easy to find inverse. Since it preserves x_1, it is easy to find inverse: just calculate $x_2 = y_2 - f_2(x_1)$; and $x_3, ..., x_n$ one by one. As a result, one may try to design a public key cryptosystem based on Tame transformation as the following:

- Private key:T, F, S, where T and S are invertible linear transformations. (Attention: when stating T and S are invertible here, it means they are invertible limited on their image set. which means, they can be only injective but not subjective.)
- Public key: P, where $P = T \circ F \circ S$.

This looks pretty nice: the two linear transformations T, S hides the vulnerable part x_1. With T, S known, one can easily solve the equation

$$y = P(x) = T \otimes F \otimes S(x) \iff P(x) = T^{-1} y S^{-1}.$$

However, with T, S unknown, it should be hard to directly solve the multivariate quadratic equation $P(x) = y$. (This is not true.)

Now the problem is: Can you break it? The answer is "yes".
(Warning: we may use row vector instead of column vector in this

Proposition 1.1. *Let $P(x) = (p_1(x), ..., p_n(x))$. Then, $\exists (a_1, ..., a_n) \neq 0$, s.t. $\sum_{i=1}^{n} a_i p_i$ is a non-trivial linear transformation.*

Proof.

$$F_1 = proj_1 \circ T^{-1} \circ P$$

is such a linear transformation, where $proj_1$ denotes taking the first element. □

You might be curious that how do we find such coefficients $(a_1, ..., a_n)$. We demonstrate using an example.

In this example, we work in GF(5). Let

$$F(x_1, x_2) = \begin{pmatrix} x_1 \\ x_2 + 2{x_1}^2 \end{pmatrix};\ S = \begin{pmatrix} 2 & 3 \\ 1 & 0 \end{pmatrix}; T = \begin{pmatrix} 1 & 2 \\ 1 & 3 \end{pmatrix}.$$

$$P(x_1, x_2) = T \circ F \circ S(x_1, x_2) = \begin{pmatrix} (2x_1 + 3x_2) + 2(x_1 + 2(2x_1 + 3x_2))^2 \\ (2x_1 + 3x_2) + 3(x_1 + 2(2x_1 + 3x_2))^2 \end{pmatrix}.$$

It is easy to see that $3p_1 - 2p_2 = 2x_1 + 3x_2$. Now, one can reduce the number of variable to 1, and directly solve the equations.

This example seemingly over-simplified the problem, but it actually captured everything we need to do here. For more complicated P (with more variables), generally we have the following equations:

$$P_1 = q_1 + l_2;$$

$$P_2 = q_2 + l_2;$$

$$...$$

$$P_n = q_n + l_n.$$

Where l_n are purely linear and q_n are purely quadratic. We already know that there is some linear transformation of P is linear:

$$A \cdot P = (a_1, ..., a_n) \cdot (P_1, ..., P_n) = linear$$

which means A will eliminate all quadratic terms in P. Then one can simply solve a linear equation for coefficients of $x_i x_j$ (so that the coefficients are all 0):

$$A \cdot (q_1, ..., q_n) = 0.$$

Though the number of equations is much larger than n, the kernel is at least 1 dimensional (by prop 4.1). It might appear a kernel with dim > 1, then one can solve the equation set up by coefficients of linear terms:

$$A \cdot l = (a_1, ..., a_n) \cdot (l_1, ..., l_n) \neq 0.$$

Combine the solution of this "equation" and the previous equation, one can get the desired solution $A = (a_1, ..., a_n)$ such that

$$(a_1, ..., a_n) \cdot (P_1, ..., P_n) = \sum_{i=1}^{n} h_i x_i; h \neq 0.$$

In another word, we have proven that we can always recover some none-zero linear combination of $x_1, ..., x_n$ from $P(x_1, ..., x_n)$.

Once this step is done, the remaining steps are easy: firstly, one reduce the number of variables of P, by simply replacing one variable x_j ($h_j \neq 0$) by linear combination of other variables.

It might not be that easy to see that after this reduction, P is still a composition of one linear transformation, one tame reduction and another linear combination. We first show this is true when $A \cdot P = x_1$ exactly. If so, simply take $x_1 \equiv 0$ in P. Since $S((0, x_2, ..., x_n)^T)$ is an invertible linear transformation of $(x_2, ..., x_n)$, it is clear that P remains similar structure. Now for general $A \cdot P$, we can always use a linear transformation (which can be absorbed into S) to make it be x_1.

Based on what we have shown, we can repeat the process of variable reduction again and again, until there is only 1 variable, and then solve the quadratic equation directly. The "Tame transformation scheme" is easily broken in polynomial time.

1.2 Minus Method

We may conclude on the failure of the above scheme: two linear transformations are not enough to "conceal" or to "hide" the linear part $F_1(x) = x_1$. One may try to fix this problem by a "Minus method": delete the first entry, or the first k entries of F, for example:

$$F(x) = \begin{pmatrix} x_3 + f_3(x_1, x_2) \\ x_4 + f_4(x_1, x_2, x_3) \\ ... \\ x_n + f_n(x_1, ..., x_n) \end{pmatrix}.$$

Then there will be no linear entry in f. Notice after this "minus method", F is no longer an injection and one would not be able to use it as an encryption scheme, but only can be used in signature system. Is this enough to ensure the security?

The answer is still a "no", but the attack method must be modified. Notice that

$$P_j = q_j + l_j, \quad j = k + 1, ..., n$$

still holds here, where q_j is a purely quadratic function of $(x_1, ..., x_j)$, i.e. a quadratic form with

rank j on $\mathbb{R}^n$. More precisely, q_j has the following form:

$$q_j = \begin{pmatrix} Q^j & 0 \\ 0 & 0 \end{pmatrix},$$

where Q^j is a $j \times j$ symmetric matrix. Hence, we will have quadratic forms of rank $k+1, k+2, ..., n$ in $F(x)$. Notice that linear transformation S will not change the rank for each quadratic form (it act on q_j as $q_j \to S^T q_j S$). Thus, if we can find a linear combination of q_j (this is a linear transformation on P and can be absorbed into T), such that the result quadratic form has a lowest rank ($\leq k+1$), then we can reduce the problem just like in the previous section by reducing the number of variables. This is known as the rank attack (or MinRank problem attack), see [2]. Diffie and Shamir had tried to fix this problem, but none of their attempt worked.

Remark: Professor had claimed this attack quite easy to figure out ourselves in the lecture, but I seriously doubt this claim.

1.3 Lifting from Finite Field

Now we introduce another construction based on multivariate quadratic equation. This idea is described by Tsutomu Matsumoto in 1988 and (sadly) broken in 1995.

Consider the following maps:

$$GF(q)^n \xrightarrow{S} GF(q)^n \xrightarrow{F} GF(q)^n \xrightarrow{T} GF(q)^n$$

Where we want S and T be linear and invertible, F quadratic and invertible. Notice that $GF(q)^n$ can be viewed as a finite field $GF(q^n)$ in the following way:

$$c = (c_0, ..., c_{n-1}) \in GF(q)^n \leftrightarrow c = c(x) = \sum_{i=0}^{n-1} c_i x_i \in GF(q)[x]/g(x) \cong GF(q^n),$$

where $g(x)$ is a degree n irreducible polynomial in $GF(q)$. In our context here, $q = 2^k$ for some k.

We construct F in the following manner: pick $\theta \in \{1, 2, ..., n-1\}$ s.t. $\gcd(q^\theta+1, q^n-1) = 1$. This ensures that

$$F' : GF(q^n) \to GF(q^n), c \mapsto c^{q^\theta+1}$$

is a 1 to 1 map (since it is a 1 to 1 map in the multiplication group $GF(q^n)*$). We claim that if we lift this map to $F : GF(q)^n \to GF(q)^n$ (or by abuse of notations, consider F' as a map

on $GF(q)^n$), then it is a quadratic map.

Proof. Since $c \mapsto c^q$ in field with $char = q$ is a field endomorphism (known as Frobenius map), $c \mapsto c^q$ is obviously an invertible linear map. Hence $c \mapsto c^{q^\theta}$ is an invertible linear map. Hence, we have:

$$F(c) = c^{q^\theta} \times c;$$

This is a quadratic map, by polynomial multiplication (all coefficients are purely quadratic terms). □

Lecture 5 ends here.

2 Lecture 6 Multivariate Cryptography (II)

2.1 Rank Attack

In this lecture, we shall first explain the rank attack performed on the encryption scheme based on some triangular maps. Recall the triangular transformation (with the first $k-1$ rows eliminated):

$$F: \begin{pmatrix} x_1 \\ ... \\ x_n \end{pmatrix} \mapsto \begin{pmatrix} x_k + f_{k-1}(x_1, ..., x_{k-1}) \\ ... \\ x_n + f_{n-1}(x_1, ..., x_{n-1}) \end{pmatrix}.$$

And denote the quadratic forms in the i-th entry of P as P_i. (We can simply drop the linear terms here.) Since the linear transformations T and S act on each quadratic terms as the following:

$$S: f_i \mapsto f_i' := S^T f_i S;$$

$$T: \begin{pmatrix} f_{k-1}' \\ ... \\ f_{n-1}' \end{pmatrix} \mapsto T \begin{pmatrix} f_{k-1}' \\ ... \\ f_{n-1}' \end{pmatrix} = \begin{pmatrix} p_{k-1} \\ ... \\ p_{n-1} \end{pmatrix}$$

It is clear that any invertible linear transformation S will not change the rank of each quadratic forms f_i, and T will only mix the $rank = k_1, ..., n-1$ quadratic forms $f_{k-1}'...f_{n-1}'$ together. Rank attack aims to find a p_i with highest rank $(n-1)$, and find a_j such that

$$Rank(a_j p_i + p_j = n-2).$$

We first show that why this strategy works, and then show how this strategy works.

Suppose we have already find such a set of $q_j = p_i + a_j p_j$ (totally $n-k$ of them). We will assume (for now) that each $q_j = a_j p_i + p_j$ is a linear combination of $f_{n-2}', ..., f_{k-1}$, which means we have eliminated the terms f_n' completely. Now, we perform rank attack on the $n-k$ quadratic forms of rank $n-2$ again the reduce the rank and size by 1. If we are lucky enough, we will finally arrive at a rank $k-1$ quadratic form which is a linear combination of f_i's. We hope this is f_{k-1}', and finding f_{k-1}' will in fact completely break the system.

A lot of problem might arise in this process.

Problem 1: Why (and How) we can find $a_j p_i + p_j$ with lower ranks?

Answer: We can assume that in general all p_i has rank $n-1$, in a probabilistic sense. For fixed p_i, p_j we try to solve the following equation:

$$rank(p_i + p_j x) \leq n-1.$$

This is equivalent to:

$$\forall M, det(M) = 0$$

where M is $n-2$ dimensional minus (n-2 维子阵) of $p_i + p_j x$. The equation $det(M) = 0$ will give a polynomial of x with $deg \leq n-1$. We will get different polynomials of x by different M, and by finding their gcd, we can find a low degree polynomial and solve it for $x = a_j$.
It might happen that the rank equation $rank(p_i + p_j x) \leq n-1$ has more than 1 solutions, and we are only interested in the only solution that eliminates f'_{n-1}.

Here is an example of undesired solution:

$$f'_{n-j} = \begin{pmatrix} I_{n-j} & 0 \\ 0 & 0 \end{pmatrix}; p_1 = f'_{n-1} + f'_{n-3}; p_2 = f'_{n-1} - 3f'_{n-3}.$$

It is clear that $q = p_1 + p_2 = 2f'_{n-1} - 2f'_{n-3}$ and $q' = p_1 - p_2 = 4f'_{n-3}$ are all solutions to $rank(p_1 + xp_2) \leq n-1$. However, we are only interested in the latter solution since it eliminates the f'_{n-1} term, but not the previous one.

(Notice that this example also shows that rank of p_i might not be $n-1$ even if it contains f'_{n-1} term.)

This can happen in the process of rank attack, however, we can always check whether a solution is "undesirable": by checking whether some linear combination of a solution and another solution will increase the rank again to $n-1$. In the previous example, one will easily find that $rank(q+q') = n-2$, which implies there must be f'_{n-1} term in q or q'. If this happens, we will discover and abandon the "undesirable solution".

Problem 2: If such problems can happen, why rank attack still works?

Answer: It still works since **in a probabilistic sense**. In cryptography, we never require our attack to work 100% of time (deterministically), we only hope it to work 20% of time, or even 2% of time and we are satisfied.

We will now give a sketch of explanation of why rank reduction attack can really complete in polynomial time, with a high probability to success. The whole attack requires $\leq n$ steps of rank reduction, and in each step, one need to solve $\leq n$ rank equations, each of them is finding gcd of $\leq n^2$ polynomials). The complexity of this process is controlled by, say, $O(n^5)$, and we only need to check that the "confirmation step" for each solutions to be "desirable" (the highest rank term has been eliminated) can complete in polynomial time with a high probability. We need to estimate the probability that our checking works. We look at the following equation:

$$rank(k(f'_{n-1} + \sum_{i=k+1}^{n-2} \alpha_i f'_i) + \sum_{i=k+1}^{n-2} \beta_i f'_i)) = n - 1.$$

If the checking does not work, then it implies that $\forall k \neq 0 \in F_q$, we always have $rank = n - 1$. Again, this situation can often happen (counterexamples are easy to construct, if we take each f'_i diagonal). The actually attack follows a backward search strategy. Instead of checking everytimes when we meet a solution (a linear combination of quadratic forms that lowers the rank), we always take some such solutions and proceed to the next step of rank reduction. If there are some "undesirable solutions" in it, then at some step, it is **very likely** that we will encounter the situation that a rank equation cannot be solved, which means at least one of the two quadratic forms is flawed as it contains higher rank term.
Now we try to explain why this process is likely to end in polynomial time.

Proposition 2.1. *Expectation of number of undesirable solution for a rank equation to yield is approximately bounded by* $\frac{1}{p-1}$*, where* p *is the size of field.*

We will not prove this statement (I haven't found a way to rigorously prove this), so we will just use it as a truth from experiments. As a result of this proposition, we can assume that after the first step of rank reduction, we will have in total around $\frac{p}{p-1}n$ solutions (forms with $rank < n-1$), with exactly $n-1$ of them are desirable solutions. What we try to do now is to estimate the time we need for discovering one of the solutions to be undesirable (which means, we have removed one undesirable solution). If this process can finish in polynomial time, then since there are in total at most $O(n)$ undesirable solutions, in polynomial time we can remove all undesirable solutions and there will be only desirable ones.

How exactly do we confirm that one solution is undesirable? Besides trying to find an x such that $rank(p_i + p_j x) < rank(p_i) = rank(p_j)$, we can also check if for any $x \in F_p$, $rank(p_i + p_j x) > rank(p_i) = rank(p_j)$. If the later happens, or the previous not stands (cannot find solution), we will immediately know that at least one of p_i, p_j is undesirable. (We have explained above.)

Proposition 2.2. *The probability for this checking to fail is strictly less than 1 and the upper bound is only related to the field size p. I guess it is also* $\frac{1}{p-1}$.

If the checking fails successively for a "depth" of a (which is to say, if a $rank = w$ solution is undesirable and we failed to discover this until we reduce the rank to $w - a$), which is an event of probability less than $(p-1)^{-a}$. In this situation, we find that in total $a+1$ $rank = n-2$ solutions can be undesirable. We can assume there are around $\frac{p}{p-1}a$ solutions that might replace such solutions (among $a+1$ $rank = n-2$ solutions there are likely to be more than $\frac{p-2}{p-1}$ of them to be "the only choice" in the rank equation and confirmed desirable, while others might be replaced by other solutions in rank equation). This makes us satisfied, since the situation we must one by one check in total less than $p^{\frac{1}{p-1}a}$ situations is only $\frac{1}{p-1}^a$, so the probability of removing an undesirable solution takes more than polynomial time is exponentially low, which makes the whole algorithm polynomial time.(Notice that this is not a probabilistic algorithm, but a deterministic one: we will always know in some step that one undesirable solution is undesirable. An analogy is, we toss a coin and hope it land with face on top, and we know whether each toss get face or tail. We hope to succeed in total around $\frac{1}{p-1}n$ times, this obviously can be done in polynomial time with good chance.) This completes the proof.

One might still doubt that the "checking" may not work, that in extreme situation, one may arrive and some form with $rank = k-1$ or even $< k-1$, but it still an undesirable solution. For example, if $f'_{n-i} = \begin{pmatrix} I_{n-i} & 0 \\ 0 & 0 \end{pmatrix}$, then $rank(f'_{n-1} - f'_{n_2}) = 1 < k-1$ and the checking will not work. Well, the interesting point here is that such a solution with low rank is exactly the solution we want, at least as much as we want to find f_{k-1}, since the low rank of any quadratic form will allow us to perform variable reduction! And when we recall our algorithm, one will find that we only stop and go backward to remove some undesirable solution only when we cannot precede to lower the rank: some rank equation has no solution at all. So we don't necessary arrive a f'_{k-1} or some f'_{k+a}–the algorithm might yield some different low rank quadratic form, which is as good as f'_{k-1} or even better!

Remark: This is not a rigorous proof: I made some unverified assumptions, and I did not estimate the probability precisely but only give an extremely loose bound. If you find better proof, please contact me.

2.2 Linearization equations

Now we will move on to field lifting scheme described in the previous lecture. We have explained how it works in the last lecture, and now we introduce a method of breaking it: Linearization Equation Method.

This method is discovered by a French mathematician Jacques Patarin in 1999. Before this surprising discovery, this encryption scheme had nearly been accepted as a encryption standard by Japan.

We recall the encryption scheme:

- Private key: T, θ, S;
- Public key: $P = \begin{pmatrix} p_1 \\ ... \\ p_n \end{pmatrix}$ Where each p_i are quadratic funtions (again, we drop linear terms here.)
- Plaintext: $x' = (x'_1, ..., x'_n)$;
- Cyphertext: $y' = p(x'_1...x'_n) = (y'_1...y'_n)$.

We will use x'_i, y'_i to denote some specific text and x_i, y_i to denote variables. Now we present the central claim of linearization equation.

For all fixed p, there exist a list of linearly independent equations:

$$\sum_{i,j=1}^{n} a_{i,j} x_i y_j = 0, \forall y = p(x).$$

If we have found a large list of such equations, we can easily recover x' from y'. It is obvious that if we find n such equations, then the solution for x' given y' is unique. Even if we only have $n - t$ such linearly independent equations, we can still list all the solutions in the kernel and try them one by one, when t is small enough. Hence, finding such linearization equations will completely break the system.

Now we are left with two problems: (I) How many such equations exist? (II) How do we find such equations? In real life, the second problem is more important for us, since if it works, it directly implies that enough such equations exist.

We shall first address the second problem, since the answer is surprisingly easy. Recall that with public key P, we can always calculate y' from any x' we want. Then, one can try to solve $(a_{i,j})$ by solving the following equation:

$$\sum_{i,j} x'_i y'_j a_{i,j} = 0$$

Given any x', y', this is a linear equation of size $\frac{n(n-1)}{2}$, and it is very likely that with enough trials for different x', y', we can find nearly all such linearization equations. (It is easy to calculate the probability of collecting all linear independent sets of $(a_{i,j})$ in the first k trials by markov chain: $E(k) = N \log N$, assuming the probability of getting each sets $(a_{i,j})$ are equal if we choose x', y' randomly.)

Now we are left with the only problem: how many such equations exist?

Firstly, we observed that their is a one to one map from linearization equations of $P = T \circ F \circ S; y = P(x)$ to linearization equations of $y = F(x)$, by the folllowing construction:

$$x^T A y = 0 \iff (Sx)^T ((S^{-1})^T A T)(T^{-1} y) = 0.$$

Hence, we only need to estimate the number of linearization equations for $F : x \mapsto y = x^{q^\theta+1}$. From this we have:

$$y^{q^\theta - 1} = x^{q^{2\theta}-1}$$

$$\iff y^{q^\theta} x - y x^{q^{2\theta}} = 0.$$

Here (by abuse of notations), x, y are in the extended field F_{q^n}, so y^{q^θ} is actually a linear transformation of y and $x^{q^{2\theta}}$ is actually a linear transformation of x. Hence, the above is actually a linearization equation, and we need to find what is its rank. Instead of counting rank, we use a classical technique: count the dimension of kernel (solution space of x). (In fact, this is what we really care about.)

We claim that for every fixed y, there are exactly q solutions for x, containing 1 trivial solution $x = 0$, when n is a prime number, which is to say that the kernel is 1-dimensional.

Proof. We only need to estimate the number of solutions for equation

$$x^{q^{2\theta-1}} = c$$

for some $c \in F^*_{q^n}$. From the construction we know at least 1 solution (x, y) exists, so we only need to prove for the upper bound, which means we can just assume $c = 1$. Then number

of solutions for $x^{q^{2\theta-1}} = 1$ is $\gcd(q^{2\theta-1}, q^n - 1)$, since the order of the multiplication group $F_{q^n}^*$ is $q^n - 1$. Notice that $\gcd(q^{\theta+1}, q^n - 1) = 1$ by definition of θ, so we only need to find $\gcd(q^{\theta-1}, q^n - 1)$. This is just $q^{\gcd(\theta,n)}$ (**The proof of this is left as homework**) and by definition n is prime, so number of non-zero solutions is exactly $q - 1$. □

Now we have completely described and proven the attack by linearization equations towards the previous encryption scheme, where Matsumoto and Imai had been defeated by Patarin. Later there are other on multivariate encryption schemes, which are mainly two types: those insist on the construction of $x^{q^{\theta-1}}$ has quadratic function of x, since it is very easy to store the private key when the quadratic map is nothing but a large number $\theta < n$. One of the ideas involving hiding the field (HFE) had appeared, and completely defeated in 2021 by Prof. Ding. Another idea is really genius: it is based on linearization equations (more or less as a private key). This method is developed by Patarin, which is called OV, Oil and Vinegar scheme; this scheme is later broken by Shamir (the "S" in RSA), and later modified to be UOV, unbalanced Oil and Vinegar scheme, which is considered very safe at least for now. We will move on to OV and UOV in the next lecture.

Lecture 6 ends here.

3 Lecture 7 Multivariate Cryptography (III)

3.1 Introduction of OV/UOV scheme

As we have mentioned in the last lecture, the encryption scheme developed by Matsumoto and Imai had been defeated. The idea of multivariate encryption had developed into two different types from this point. One type is based on the field extension construction (similar to M&I scheme), which is called the HFE (Hidden Field Encryption). This method has been completely defeated in 2021 by Ding's team. The other type is the Oil and Vinegar scheme (later developed to unbalanced OV), which is an encryption scheme converted from the linearization attack. We will now introduce Ov scheme.

OV (Oil and Vinegar) is a signature scheme. It map the space F_q^{o+v} to space F_q^o. Denote $x \in F_q^{o+v}$ as $x = (x_1, x_2, ..., x_o, x_1', x_2', ..., x_v')$.

$$P = F \circ S : F_q^{o+v} \to F_q^o,$$

where S is a bijective linear transformation of x (that mixes the variables) and $F = \begin{pmatrix} F_1 \\ ... \\ F_o \end{pmatrix}$ o polynomials, and

$$F_k = \sum_{i,j} a_{i,j}^k x_i x_j' + \sum_{i,j} x_i' x_j' + l_k(x)$$

$$= \sum_{i=1}^{o} \sum_{j=1}^{v} a_{i,j}^k x_i x_j' + \sum_{i=1}^{v} \sum_{j=1}^{v} b_{i,j}^k x_i' x_j' + l_k(x),$$

where $l_k(x)$ is the linear term, and coefficients of quadratic terms $a_{i,j}, b_{i,j}$ are randomly chosen with a uniform distribution in F_q. It is easy to understand why this encryption scheme is called "Oil and Vinegar": in the quadratic terms, there are only oil mixed with vinegar and vinegar mixed with vinegar, but not oil mixed with oil. (This has some analogy to cooking.)

Here, if we choose the parameter $o = v$, then the system is called "balanced", which has been broken by Shamir. For $v > o$, the system is called unbalanced and considered very safe till nowadays.

In this signature scheme, the public scheme is P, and the private key is F, S. It seems difficult to calculate the "inverse" for P (here, inverse means find at least one pre-image): the quadratic polynomials F_k (or one can view them as quadratic forms) are not easy to find inverse directly, as they are **really randomly chosen**. However, one can utilize the linearization

equations to find inverse, if he knows the secret key S.

How to set up the linearization equations? The basic idea is that, since F maps F_q^{o+v} to F_q^o, it is very likely that for any set of $x' = (x'_1, ..., x'_v)$, there will be a pre-image x with vinegar terms just equal to x'. So one can just randomly choose some $x' = (x'_1, ..., x'_v)$, and get the following k linear equations of $(x_1, ..., x_o)$: (here c_k is the term determined by the image of F_k; $k = 1, 2, ..., o$.)

$$\sum_{i=1}^{o}\sum_{j=1}^{v} a_{i,j}^k x'_j x_i = -\sum_{i,j=1}^{v} b_{i,j}^k x'_i x'_j + c_k.$$

(By abusing notations, here the oil $(x_1, ..., x_o)$ and vinegar $(x'_1, ..., x'_v)$ parts has undergone the linear transformation S. It is clear that if one does not know the linear transformation S, these linear equations can never be set up since coefficients $a_{i,j}^k, b_{i,j}^k$ are unknown; also, he will not be able to tell the difference between O and V.)

If the linear equation on the LHS is solvable, i.e. the coefficient matrix

$$A = (\sum_{j=1}^{v} a_{i,j}^k x'_j)_{1\le i,k\le o}$$

is invertible, then one set of $x_1, ..., x_o$ correspond to x' can be directly solved. This will yield a pre-image x with respect to the image given. (This holds for both balanced and unbalanced OV.)

For the efficiency of this algorithm, we will not provide a rigorous proof here, but we should mention one fact:

$$|GL_n(F_q)| = \prod_{k=0}^{n-1}(q^n - q^k);$$

$$\frac{|GL_n(F_q)|}{|M_n(F_q)|} = \prod_{k=1}^{n}(1 - q^{-k}).$$

The last (very close to 1) value can be viewed as the probablity that A is invertible if A is uniformly distributed on $M_o(F_q)$ upon the choice of x'. In practice, this algorithm seldom requires more than three times of trials of x'. This explains why the decryption of OV/UOV works.

3.2 Defeating OV

Now, we shall introduce a method to break OV (but not UOV).

Consider the polynomial F_k as quadratic forms (symmetric matrix):

$$F_k(x) = (x'_1, ..., x'_v, x_1, ..., x_o) \begin{pmatrix} * & * \\ * & 0 \end{pmatrix} (x'_1, ..., x'_v, x_1, ..., x_o)^T.$$

(For the sake of convenience, we write x as $(x'_1, ..., x_o)^T$ here.) And P_k (the k-th element of P) is given by:

$$x^T S^T A_k S x, A_k = \begin{pmatrix} * & * \\ * & 0 \end{pmatrix}.$$

So we can just denote P_k as $S^T A_k S$. Here comes the most important observation:

$$A_k : O \to V.$$

What does this mean? By matrix multiplication, one can see that

$$\begin{pmatrix} * & * \\ * & 0 \end{pmatrix} \begin{pmatrix} 0 \\ \vdots \\ 0 \\ * \\ \vdots \\ * \end{pmatrix} = \begin{pmatrix} * \\ \vdots \\ * \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

This implies that if we view O and V as two subspaces of F_q^{o+v}, we have $O^\perp = V$; $\forall x \in O, A_k x \in V$. Suppose there is at least two A_ks ($k = 1, ..., o$) are invertible (this is very likely to be true) , then for some $i \neq j$:

$$A_j^{-1} A_i : O \to O;$$

$$P_j^{-1} P_i = S^{-1} A_j^{-1} S^{T^{-1}} S^T A_i S = S^{-1} A_j^{-1} A_i S : S^{-1}(O) \to S^{-1}(O)$$

This suggest that the matrix $A_j^{-1} A_i$ has the block form:

$$\begin{pmatrix} K & 0 \\ * & K^T \end{pmatrix}$$

If it is not so obvious, we can see it from the block form of A_j^{-1} and A_i:

$$A_j^{-1} A_i = \begin{pmatrix} 0 & C \\ C^T & D \end{pmatrix} \begin{pmatrix} A & B \\ B^T & 0 \end{pmatrix} = \begin{pmatrix} CB^T & 0 \\ * & BC^T \end{pmatrix}$$

For the sake of convenience, denote $A_j^{-1} A_i$ as A. Let f be the characteristic polynomial of A: $f(\lambda) = det(\lambda I - A)$, such that $f(A) = 0$. From the block form of A, it is clear that:

$$f = g^2,$$

where g is the characteristic polynomial of K (characteristic polynomial is invariant under transpose). Somehow g is highly likely to be irreducible (you can find it by experiment). And we have:

$$g(A) = \begin{pmatrix} 0 & 0 \\ L & 0 \end{pmatrix}$$

Where L is a $o \times o$ matrix; we can assume that it is invertible with a high probability. Then the kernel of $g(A)$ is exactly the O space, and the image is the V space.(If L is not full-ranked, the kernel will be larger but contain the whole O space, but one can still pick some vectors in it with high probability that it is inside O.)

Now we can describe the "Shamir Attack" to break OV. Firstly, pick some invertible P_i, P_j and calculate $P_j^{-1}P_i := P_{ij}$. And then calculate the characteristic polynomial $f(\lambda) = det(\lambda I - P_{ij})$. Notice that $P_j^{-1}P_i = S^{-1}A_j^{-1}A_iS$, so the characteristic polynomial of $P_j^{-1}P_i$ is the same to $A_j^{-1}A_i$. Then, factorize f to get $g = \sqrt{f}$; $g(P_j^{-1}P_i) = S^{-1}g(A_j^{-1}A_i)S$. The kernel of $g(P_j^{-1}P_i)$ is just $S^{-1}(O)$, where $O = span(e_{o+1}, e_{o+2}, ..., e_{2o})$ (for OV, $o = v$); the image of $g(P_j^{-1}P_i)$ is $S^{-1}(V)$, where $V = span(e_1, ..., e_o)$. So the kernel gives the matrix:

$$ker = S^{-1}\begin{pmatrix} 0 & 0 \\ 0 & I_o \end{pmatrix};$$

and Image gives the matrix

$$Im = S^{-1}\begin{pmatrix} I_v & 0 \\ 0 & 0 \end{pmatrix}.$$

Combining these two we will get the private key S, and it is easy to find F from S, P.

Remark: One might be curious about the complexity of factorizing the polynomial f on F_q. Generally, by Berlakamp-Massey algorithm, time complexity for factorizing deg n polynomial on F_q is $O(n^2)$. (See Berlekamp-Massey).

3.3 UOV

For UOV (unbalanced Oil and Vinegar), the parameter o, v are chosen such that $v > o$, usually $v = \frac{3}{2}o$ or $2o$.Let's take $v = 2o$. The consequence of this is immediate:

$$dim(A_k(V)) = 2o;$$

$$A_k(V) \subsetneqq O.$$

If A_k invertible. So $A_j^{-1}A_i$ maps O to some subspace with $dimv$, and Shamir attack no longer works. (The probability of finding at least 1 vector $\in$ O subspace is only $Pr \leq q^{o-v}$.)

Moreover, we discovered that UOV looks like any quadratic polynomial if you try to solve it directly using Gröbner basis method, i.e. it does not show any traits.

Before we introduce Rainbow signature scheme, we should first talk about some general methods to solve quadratic polynomials on F_q, given the condition that the solution is unique.

One basic method is the XL (eXtended Linearization) method. This method can be seen as some kind of Gr¨obner bases method: we consider the set of multivariate quadratic polynomials generates an ideal of the polynomial ring $R = F_q[x_1, ..., x_q]/(x_i^q - x_i)$. Let I be the ideal, since the condition that the solution exist and is unique is given, we have:

$$I =< f_1(x_1, ..., x_n), ..., f_n(x_1, ..., x_n) >=< (x_1 - a_1), ..., (x_n - a_n) >$$

and what we will do is actually trying to find such set of generators for I from the original set of polynomials. It is clear that $\exists g_1, ..., g_n \in R$. s.t.:

$$g_1 f_1 + ... + g_n f_n = x_1 - a_1.$$

If we can find such set of $g_i f_i$, we can solve for some x_k and reduce the number of variables. For the classical XL method, we multiply f_i by linear terms ($deg = 1$ polynomials) and get $deg = 3$ polynomials; we list all the linear combinations of these polynomials and check whether some univariate polynomial is generated. If yes, we can solve it; if no, we further increase the deg by multiplying linear terms and get $deg = 4$ polynomials, and try to reduce to univariate by linear combination. This process will always ends at $deg = q$ since the ring consists of polynomials with $deg \leq q$.

We should emphasize that this method is not efficient in general: too many polynomials must be stored, and it is not polynomial time in most cases.

Somethimes, the "ceiling" (max deg of polynomials) of XL method could be lowered. For example, in M&I scheme, since linearization equations exists:

$$\sum a_{ij} x_i y_j = 0;$$

And y_j are all quadratic w.r.t. x. In this case, all $deg = 3$ polynomial will collapse, and revised XL method (called mutant XL) only need to deal with $deg = 2$ polynomials.

Other well-known algorithms to solve multivariate equations include F_4 algorithm from Gröbner basis (in Magma). For more information about XL and mutant XL, see [2].

Lecture 7 ends here.

4 Lecture 8 Multivariate Cryptography (iv)

4.1 Rainbow signature

In this lecture, we will introduce the Rainbow signature scheme, which is a "multilayer" version of UOV. This signature scheme is proposed by Ding and Schmit. We will only present a 2-layer version here, which can present the core idea of this signature scheme. (For more general and original design, see [1], page 116.)

In short, Rainbow signature scheme has similar encryption function and public/private key with UOV:

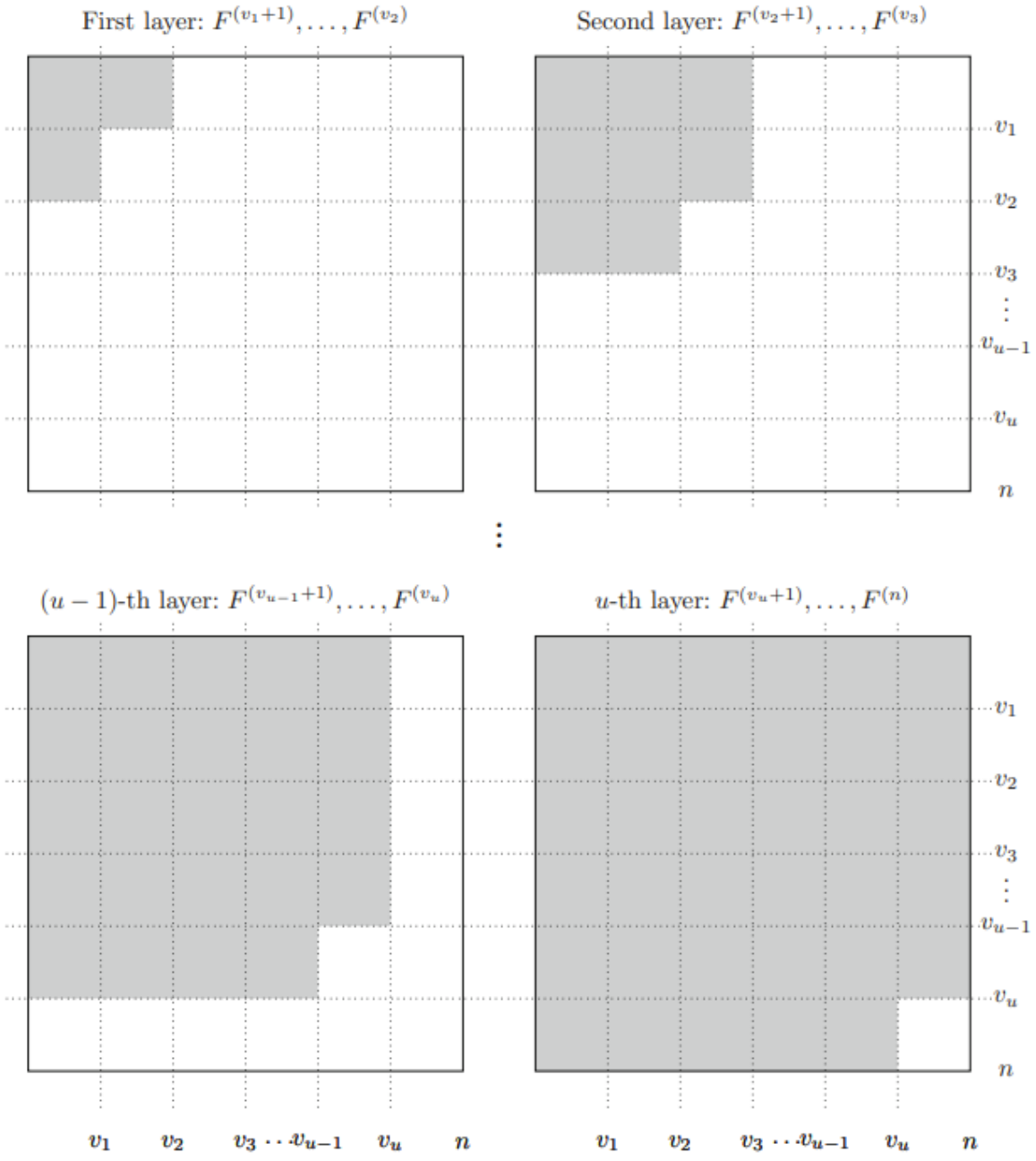
$$P = T \circ F \circ S.$$

But the quadratic map F consists of two layers (it is easy to write in block matrix form):

$$F_1 ... F_{o_1} = \begin{pmatrix} * & * & 0 \\ * & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}; F_{o_1+1} ... F_{o_1+o_2} = \begin{pmatrix} * & * & * \\ * & * & * \\ * & * & 0 \end{pmatrix}$$

Where the block size is $(v_1 + o_1 + o_2) \times (v_1 + o_1 + o_2)$. S mixes the two Oils and one Vinegar part; T mixes F_ks with different rank as matrices.

The advantage of Rainbow signature scheme is clear: the size of private key is much smaller than UOV, since more blocks of F_i are zero matrices in Rainbow. (In the original design of Rainbow, there are more layers in F, so that the quadratic forms should look like the following picture from [1]:



It is clear that a lot of F_k only have a small number of non-zero blocks, so they can be stored in smaller space.

4.2 Minrank Problem and Attacks

However, Rainbow signature scheme is vulnerable against some attacks. People have found in the last decade that it is difficult to hide the lower rank parts of F. Solving the Minrank problem appeared here will reduce the complexity of Rainbow to its lowest ranked F_k, and

consequently breaking the system if the rank of F_k is too low. (To reduce the private key size, $F_1, .., F_{o_1}$ should have very low rank.)

We will now formulate the Rank problem in Rainbow: Find $a_1, ..., a_n \in F_q$, such that

$$Rank(\sum_{i=1}^{n} a_i P_i) = o_1 + v_1.$$

Or the general Minrank problem: Given $n \times n$ (full-rank) matrices $M_1, ..., M_m$, and some constant r, find $a_1, ..., a_n$ (given the condition that the unique solution exists) such that

$$Rank(\sum_{i=1}^{m} a_i M_i = r.$$

It seems counter-intuitive, but the fact is, when r is close to n, the Minrank problem is extremely hard to solve in polynomial time; but when r gets very small, there are various efficient ways to tackle it. We will later see why. There are three ways of attack to solve Minrank problem.

The first one is know as Kipnis-Shamir attack. Suppose the low rank matrix

$$M' = \sum_{i}^{m} x_i M_i, rank(M') = r;$$

The kernel of M should have dim $n - r$. So the basis of kernel space should have the following form:

$$K = \begin{pmatrix} I_{n-r} & K^* \end{pmatrix},$$

Where $K^* = (k_{ij})$ is a $(n - r) \times r$ matrix with the following equation satisfied:

$$M' \cdot K^T = 0$$

We will now try to solve this equation (with the $(n - r) \times r$ elements in K^* and $x_1, ..., x_m$ as $(n - r) \times r + m$ variables). In general, we can use XL method, trying to eliminate the cross terms w.r.t x_i by multiplying k_{ij} again and again. This is not very efficient and Rainbow can resist this kind of attack.

The second method is the Minor method. This method aims to solve the determinant equations emerged from rank equation directly. Since $rank(M') = r$, all $(r + 1) \times (r + 1)$ Minors $= 0$. ($(r + 1$ 维子阵行列式为零). WLOG, suppose $x_1 = 1$ (if $X_1 \neq 0$, we can always rescale $x_1, ..., x_m$ such that $x_1 = 1$). There are $m - 1$ variables, and the number of monomials

$x_2^{a_2}...x_m^{a_m}$ is approximately $\binom{n+r}{r}$, and the number of equations (=number of $(r+1)\times(r+1)$ Minors) is $(\binom{n}{r+1})^2$. If $r << n$, then number of monomials « number of equations. We can try to eliminate the higher order terms through guassian elimination among coefficient matrices. however, if $r \approx n$, this method seems not efficient (NP hard). Since r is not so small in Rainbow, Minor is not applicable.

Finally, we will introduce the Supported Minor method, which works efficiently in Rainbow. Denote $M' = \sum_{i=1}^{m} x_i M_i$ be the matrix with $rank = r$. This method is similar to Minor method, but we consider the basis of the image of M' instead of the kernel. After row Gaussian elimination (and deleting the zero vectors), we can write the basis of image as the following matrix (each row vector belongs to the image):

$$\begin{pmatrix} I_r & K \end{pmatrix} := R$$

Where $K = (y_{i,j})_{1\le i\le r, 1\le j\le n-r}$. Let

$$M' = \begin{pmatrix} l_1 \\ \vdots \\ l_n \end{pmatrix},$$

where l_k is the k-th row vector of M'. Then $L_i := \begin{pmatrix} l_i \\ R \end{pmatrix}$ is a $(r+1)\times n$ matrix with $rank = r$, since all rows of M' is a linear combination of rows of R. Thus all $(r+1)\times(r+1)$ minors of L_i is 0. We will set up equation based on these minors, but not directly. Denote the $r\times r$ minors of R as R_k (where R_1 is the determinant of I_r, which is just 1); then any $(r+1)\times(r+1)$ minor of L_i is some linear combination of $y_{ij}R_k$, and again, y_{ij} is nothing but some linear combination of $x_1, ..., x_m$. As we are trying to solving for x, what we are doing here is that we use $x_1, ..., x_m$ and R_k $(k = 2, 3, ..., \binom{n}{r})$ as variables to set up some equations in the form:

$$\sum_{i,k} c_{i,k} x_i R_k = 0.$$

The total number of $(r+1)\times(r+1)$ Minors in L_i is $\binom{n}{r+1}$, so there are in total $n\binom{n}{r+1}$ equations generated by $L_1, ..., L_n$. However, there are only $\binom{n}{r} - 1 + m$ variables $R_2, ..., R_{\binom{n}{r}}; x_1, ..., x_m$ (Usually $m = n$). (Notice that $R_1 = 1$ here, so the equations are not homogeneous and one should be able to eliminate the cross-terms $x_i R_k$.) As number of equations is much larger than the number of variables when r is not so large, it is efficient to solve x by XL.

One may question whether this method can really work, since choosing R_k but not y_{ij} to be variables seems suspicious, as they are not 1-1 correspondent and there might be some

values of R_k that cannot correspond to any set of (y_{ij}). However, this does not matter since we only care about the numerical solutions of $x_1, ..., x_m$. If we review the XL method carefully, we will find that in the whole process the variables R_k only appear as formal variable but will not be taken any value.

Remark: In practice, if we use this attack on Rainbow, we might only multiply these equations with x_i but not R_k in XL method, which works well enough.

Lecture 8 ends here. (In fact, in the last 20 minutes of the lecture, HFE scheme invented by Patarin had been introduced and Kipnis-Shamir attack briefly sketched. However, since in the next lecture HFE and K-S attack was introduced in detail, we will leave it to the next lecture's note.)