katagaitai workshop 2017 winter

0CTF Finals: Authentication & Secrecy

Shiho Midorikawa

Introduction

Who

- 緑川 志穂 (Shiho Midorikawa)
  - Twitter: `@elliptic_shiho`
- Cybozu Labs, Inc.
- CTF, Math, Crypto, ...
  - 公開鍵暗号がメイン
- CTF は scryptos, binja, vuls, fuzzi3 など

今日の流れ

- このスライドでは主に **Coppersmith's Method** について解説します.
- 内容としては Medium Hard ～ Hard で出てくる問題に多いです

今日の流れ

前半:

- 基礎事項
- Coppersmith's Method
  - Howgrave-Graham's Lemma
  - 方程式を増やす
  - 組み合わせる
    - ★ 格子
    - ★ LLL
  - アルゴリズム
  - 多変数への拡張, 実装解説

今日の流れ

後半:

- Authentication & Secrecy (0CTF 2017 Finals)
  - Dual RSA
  - 攻撃手法解説
  - 実装
- Workshop

謝辞

- 本スライドを書くにあたり, katagaitai 勉強会運営の@bata_24 氏, @trmr105 氏にはレビューも含め大変お世話になった.
- また, @Bono_iPad 氏, サイボウズ・ラボ株式会社の各位にもレビュー頂き, 重要なご指摘を頂いた.
- 加えて, 今回取り上げる問題の作問者である@sea0breeze 氏, 勉強会全体のサポートをしていただいている NRI セキュアテクノロジーズ様をはじめとする多くの方々にお世話になった.
- この場を借りて感謝の意を表したい.

基礎事項

RSA

- RSA 暗号 [RSA78]

Definition 1 (RSA 公開/秘密鍵)

ある素数 p, q について $n = pq$ とおき, $\phi(n) = (p-1)(q-1)$ を割りきらない正の整数 e について $d = e^{-1} \mod \phi(n)$ とおく. このとき, (e, n) を RSA 公開鍵, d を RSA 秘密鍵という.

Definition 2 (RSA 暗号化)

RSA 公開鍵 (e, n) と平文 $m \in \mathbb{Z}/n\mathbb{Z}$ について暗号文 c は $c = m^e \mod n$.

Definition 3 (RSA 復号)

RSA 公開鍵 (e, n) と RSA 秘密鍵 d, 暗号文 c について平文 m は $m = c^d \mod n$.

RSA

- 正しく復号できることの検証
  - $m' = c^d \mod n$ とおくと,

$$\begin{aligned} m' &\equiv m^{ed} \\ &\equiv m \cdot m^{k\phi(n)} \qquad \text{where } k \in \mathbb{Z} \\ &\equiv m \cdot 1 \\ &\equiv m \pmod{n}. \end{aligned}$$

  - 途中, Euler's Thm.($\gcd(a, n) = 1$ な a について $a^{\phi(n)} \equiv 1 \pmod{n}$) を使った.

RSA

- RSA 暗号に対する暗号解析の結果は多数存在する
  - よくまとまっているサーベイとしては少し古いが [Bon99] が鉄板.
- Coppersmith's Method 関連で多大な業績を挙げている Alexander May によるサーベイ [Ale07], May の博士論文 [May03] も良い資料であるため, このスライドを読んだ後にぜひとも読んでいただきたい.
  - 方向性の似ている日本語の資料としては [KO08] がよい. いずれも本スライドを書くにあたり大いに参考にさせていただいた.
- この中でも触れられている Coppersmith's Method というものがどんなものかについて軽く触れる.

RSA

- Coppersmith's Method は既知の自然数 M を法とした方程式の小さな解を探すアルゴリズム
- 小さい解という条件だけでどのようなことができるのか?

RSA

- 例 1: Stereotyped-Message Attack
  - 暗号文 c に対応する平文 m の大部分がわかっているが, 一部だけがわからないときに使える攻撃手法
  - m が定数 C と不定変数 x_0 を用いて $m = C + x_0$ と表せるとき, 次のような方程式を立てられる.

$$(C + x_0)^e - c \equiv 0 \pmod{N}$$

- この方程式の解は, x_0 が小さいならば Coppersmith's Method を用いて得られる.

RSA

- 例 2: Boneh-Durfee's Attack
- RSA のパラメータ e, d の間には次の方程式が成立している. ただし k はある整数.

$$ed = 1 + k(p-1)(q-1)$$

この方程式を Coppersmith's Method を適用しやすい形 (なるべく線形, 変数の数/解の大きさは小さく取る) に変形する.

$$\begin{aligned} ed &= 1 + k(p-1)(q-1) \\ &= 1 + k(pq + 1 - (p+q)) \end{aligned}$$

$x = k$, $y = -(p+q)$, $A = N + 1$ とおき, e で剰余を取って

$$x(A+y) + 1 \equiv 0 \pmod{e}.$$

- $\bmod e$ によって d が消えていることに注意.

RSA

- このとき, x, y がそれぞれ十分小さければ Coppersmith's Method を適用可能.
  - この式が解けてしまえば, RSA 秘密鍵が出せる.
  - 詳細は論文 [BD99]
- このように, 最近では初心者向け問題にもなっているような問題の裏側でも用いられている重要な手法
  - 最近 ROCA[NSS+17] があったように, まだまだ最新の暗号解析の理論の中で使われている手法でもある.

Coppersmith's Method

Coppersmith's Method

- 論文によって Coppersmith's Method の定義が異なっている場合もあり, 読みにくさに拍車をかけている
  - そのため, ここで一度整理しておく.

Coppersmith's Method

- 1996 年に Don Coppersmith は EUROCRYPTO 1996 に 3 本の論文を第一著者として通している
  - Low-Exponent RSA with Related Messages [CFPR96]
  - Finding a Small Root of a Univariate Modular Equation [Cop96b]
  - Finding a Small Root of a Bivariate Integer Equation; Factoring with High Bits Known [Cop96a]
- このうち現在 Coppersmith's Method と呼ばれる手法の系譜の祖となった手法は [Cop96b] で提案された.
  - ここで提案された手法は回りくどい証明・アルゴリズムだったため, 1997 年に Nicholas Howgrave-Graham により改良されたアルゴリズムが提案された [HG97].

Coppersmith's Method

- 中でも [HG97] では Coppersmith's Method 全体の流れを決める重要な補題が提案された (後述).
  - この提案以降, Coppersmith's Method の流れをくむほとんどの手法ではこの補題を用いて示していることからも重要性は伺える.

Coppersmith's Method

Theorem 4 (Coppersmith, [HG97])

N を既知の整数とし, $f(x)$ をモニック[a] な 1 変数 δ 次多項式とする. このとき, $f(x_0) \equiv 0 \pmod{N}$ と次の条件を満たすような x_0 を効率よく求めることができる:

$$|x_0| \leq N^{\frac{1}{\delta}}.$$

[a]最高次係数が 1 であるような多項式

- [HG97] で用いられたアルゴリズムや考え方, 命題はその後の Coppersmith's Method 関連の理論研究に大きく寄与した.
  - 中でも Howgrave-Graham's Lemma と呼ばれる補題は存在意義が大きいため, この後最初に取り上げる.

Coppersmith's Method

Theorem 5 (Coppersmith, [HG01])

N を既知の整数, $0 < \beta \leq 1$ を実数とし, $f(x)$ をモニックな *1* 変数 *1* 次多項式とする. このとき, 未知の $b \geq N^{\beta}$, $b|N$ について $f(x_0) \equiv 0 \pmod{b}$ が成立, かつ次の条件を満たすような x_0 を効率よく求めることができる:

$$|x_0| \leq N^{\beta^2}.$$

- [HG01] で発表された Thm. 4 の応用.
- 方程式の法が未知の値 b であっても, その倍数 $N = k \times b$ $(k \in \mathbb{N})$ が既知であれば解くことが可能という主張.

Coppersmith's Method

Theorem 6 (Coppersmith, [May03])

N を既知の整数, $0 < \beta \leq 1$ を実数とし, $f(x)$ をモニックな 1 変数 δ 次多項式とする. このとき, 未知の $b \geq N^{\beta}$, $b|N$ について $f(x_0) \equiv 0 \pmod{b}$ が成立, かつ次の条件を満たすような x_0 を $\log N$, δ の多項式時間で求めることができる:

$$|x_0| \leq N^{\frac{\beta^2}{\delta}}.$$

- 最終形態

Coppersmith's Method

- 相違点を表にまとめると, 以下のようになる.
  - ただし, M は既知の自然数, $b < N^{\beta}$ は M の未知の因数.

提案	β	δ	方程式の形
[Cop96b], [HG97]	1	制約なし	$f(x) \equiv 0 \pmod{M}$
[HG01]	制約なし	1	$f(x) \equiv 0 \pmod{b}$
[May03]	制約なし	制約なし	$f(x) \equiv 0 \pmod{b}$

- 最後の May の手法が最も一般化されていることがわかる.
  - 実際, Sagemath の `small_roots` 関数や Pari/GP の `zncoppersmith` 関数に採用されているアルゴリズムは May のものである. このことからも, 現在標準的な (1 変数 Modular 方程式に対する)Coppersmith's Method はこの May の手法 [May03] であるといえる.

Coppersmith's Method

- ここからは Coppersmith's Method のアルゴリズムの解説を行う.
  - Thm. 6 は Thm. 4 を示せばすぐに示せるため, まずは Thm. 4 を示すことを目標とする.
- Coppersmith's Method のアルゴリズムは大まかにわけて次の 3 ステップにわけられる.
  - 方程式を増やすパート
  - 方程式を組み合わせるパート
  - 方程式を変換・解くパート

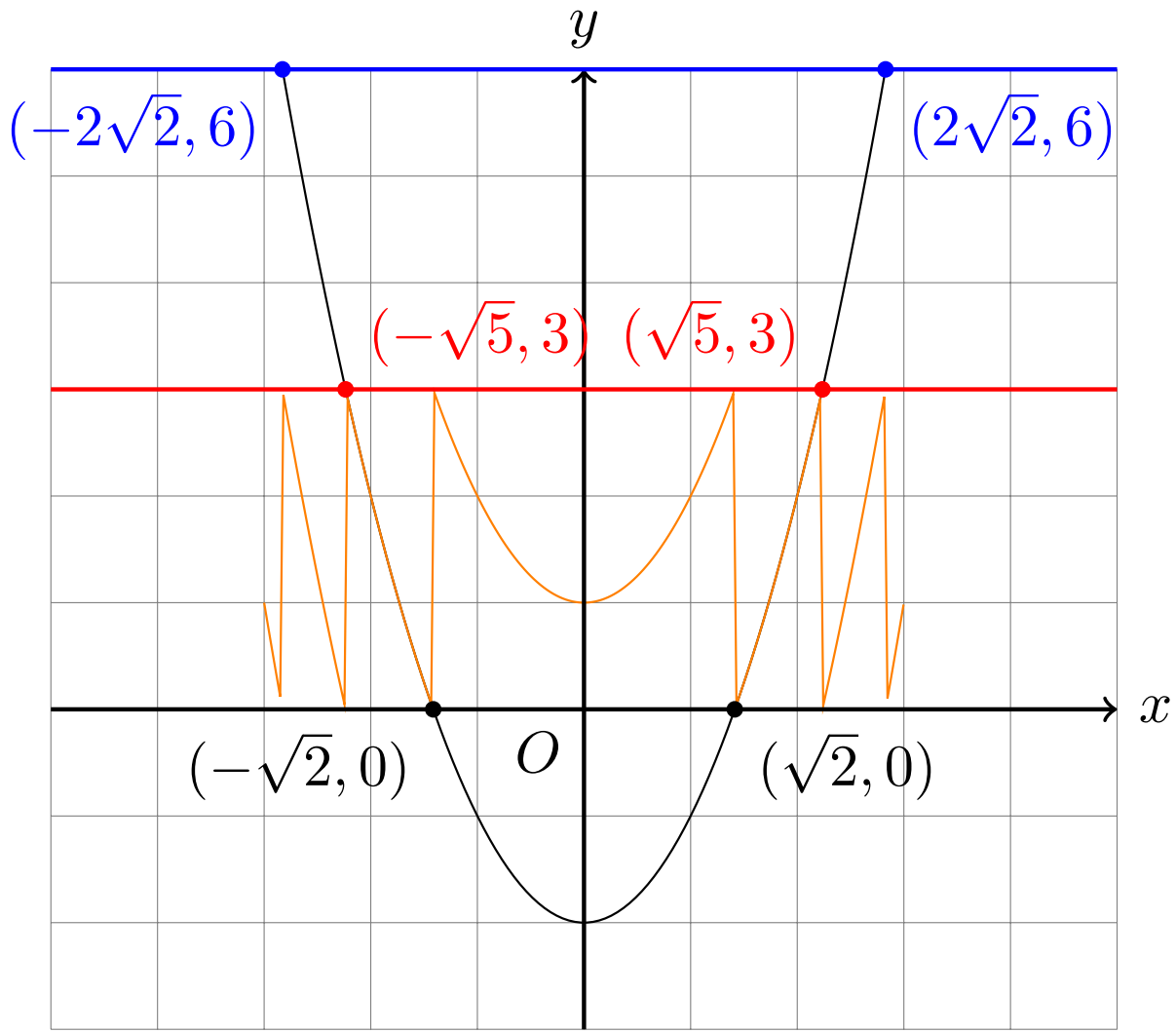
Coppersmith's Method

- 方程式を増やすパートでは, 解きたい方程式と同じ解を持つ方程式を幾つか構成する
  - イメージ的には連立方程式の方程式を式変形で生み出しているのに近い
- 方程式を組み合わせるパートでは, 前パートで構成した方程式を組み合わせ, ある条件を満たす方程式を作ることを考える.
  - この条件について, 次ページ以降考察する.
- 最後に, 方程式を変換・解くパートにおいては前パートで条件を満たすように作った方程式から実際に解を得る.

Coppersmith's Method

- まず, 考え方として Coppersmith's Method は「Modular 方程式を整数方程式に変換する」ことで Modular 方程式を解く.
  - 整数方程式に対する解法は Berlekamp 法や二分探索法等, 既に効率的な手法がいくつも存在する.
- しかし, Modular 方程式の場合
  整数方程式には存在しない解が複数存在する.

Coppersmith's Method

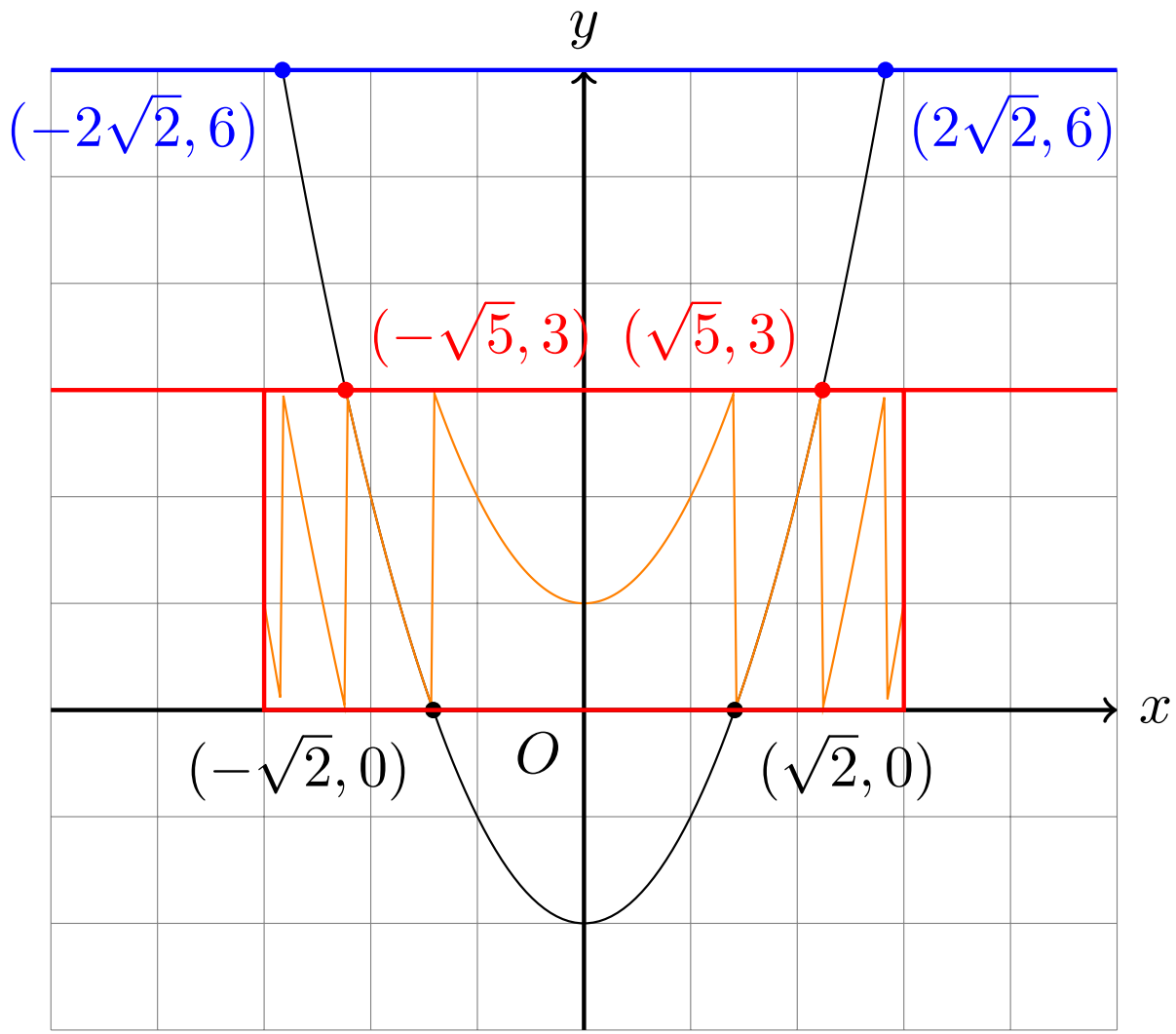


この図は, 2 次関数 $y = x^2 - 2$ と, mod 3 における同じ関数, そしてその解を図示したものである.

Coppersmith's Method

- 注意点: 説明のために連続したカーブを描くような図形を選んでいるが, 実際には解は整数しかとりえない.
  - あくまでこれはイメージと考えてほしい.
- また, Modular 方程式の解 x_0 は法 M について $|x_0| < M$ と仮定できるため, Modular 方程式の定義域は $0 \leq |x| < 3$ としている.

Coppersmith's Method



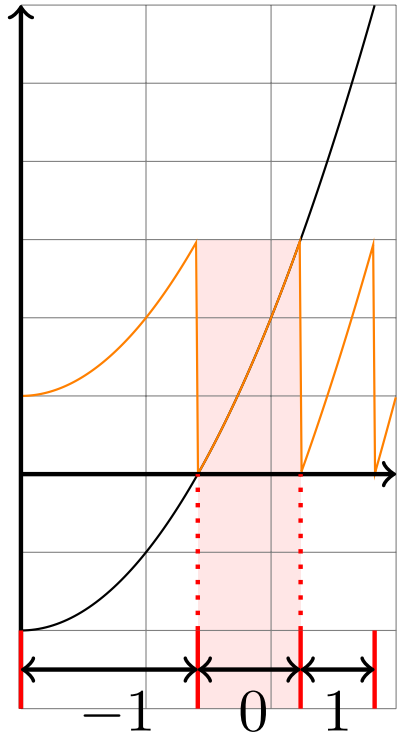
$0 \leq |x| < 3$ の間しか動くことはできないので, 解はこの 6 個で全て.

Coppersmith's Method

- 一般に, Modular 方程式の解の個数は整数方程式の個数とは異なる.
  - しかし, 先ほどのグラフをよく見ると, 整数方程式とグラフがかぶっているところがあることに気づく.
- この部分に解が存在すれば, 同じ方程式でも Modular 方程式ではなく整数方程式として解くことができるのではないか?

Coppersmith's Method

- 図形で見た事実を数式で表現する.
  - 解がグラフでいうかぶっている部分であると仮定し, 条件を導出する.
- まず, 解 x_0 を持つような Modular 方程式 $f(x) \equiv 0 \pmod{M}$ について, $k \in \mathbb{Z}$ が存在し, $f(x_0) - kM = 0$ と表せる.
  - 先ほどの「グラフが整数方程式とかぶっているところ」は式で表すと $k = 0$ である部分といえる.



Coppersmith's Method

- すると $f(x_0) = 0 \times M = 0$. これは確かに整数方程式 $f(x) = 0$ と被っている部分に解が存在することを表している.
  - 逆に $f(x_0) = 0$ ならば, $k = 0$ ととることで $f(x_0) \equiv 0 \pmod{M}$ という Modular 方程式でいいかえることができる.
- このことから, Modular 方程式が $k = 0$ で成立することと整数方程式 $f(x) = 0$ が成立することは同値といえる.
  - すなわち, 多項式 $f(x)$ に変形を加え, $k = 0$ になるように変形できればそれは整数方程式に変換したことになるといえる.

Howgrave-Graham's Lemma

- この $k = 0$ になる条件をわかりやすい基準として示したものが **Howgrave-Graham's Lemma** である.

Lemma 7 (Howgrave-Graham, [HG97])

M を法, $g(x) \in \mathbb{Z}[x]$ を整数多項式とし, 含まれる単項式の数を ω とする. $g(x)$ に対してある X が存在し, $g(x_0) \equiv 0 \pmod{M}$ なる $x_0 \in \mathbb{Z}$ について $|x_0| \leq X$ であると仮定する. このとき,

$$\|g(xX)\| < \frac{M}{\sqrt{\omega}}$$

が成立するならば $g(x_0) = 0$ が (整数方程式として) 成立する.

ただし, $\|g(x)\| = \left\| \sum_{i=0}^{\deg g(x)} g_i \right\| = \sqrt{\sum_{i=0}^{\deg g(x)} g_i^2}$ であり, $\deg g(x)$ は $g(x)$ の次数を表す.

Howgrave-Graham's Lemma

- 証明
- 仮定 $\|g(xX)\| < \frac{M}{\sqrt{\omega}}$ の元で式を変形すると以下のようになる.

$$
\begin{aligned}
|g(x_0)| &= \left| \sum_{i=0}^{\deg g(x_0)} g_i x^i \right| \\
&\leq \sum_i |g_i x_0^i| \\
&\leq \sum_i |g_i| X^i
\end{aligned}
$$

Howgrave-Graham's Lemma

- Cauchy-Schwarz の不等式の変形 $\sum_i x_i y_i \leq \sqrt{(\sum_i x_i^2)}\sqrt{(\sum_i y_i^2)}$ を用いて

$$\sum_i |g_i| X^i = \sum_i (1 \cdot |g_i| X^i)$$

$$\leq \sqrt{\sum_{i=0, g_i \neq 0} 1} \sqrt{\sum_i (|g_i| X^i)^2}$$

$g_i \neq 0$ な g_i の個数は ω で表されるので,

$$\sqrt{\sum_{i=0, g_i \neq 0} 1} \sqrt{\sum_i (|g_i| X^i)^2} = \sqrt{\omega} \|g(xX)\| < M.$$

Howgrave-Graham's Lemma

- ここで, $g(x_0) \equiv 0 \pmod{M}$ よりある $k \in \mathbb{Z}$ が存在して $g(x_0) = kM$ と書くことができる.
  - この k は $\|g(xX)\| < M$ より明らかに 0 である. このことは $g(x_0) = 0$, すなわち $g(x) = 0$ という整数方程式が解として x_0 を持ちえることを表している (証明終)

Howgrave-Graham's Lemma

- Lemma 7 は「Modular 方程式の係数が十分小さい, かつ解 x_0 が条件を満たしている場合に $k = 0$ の範囲内に解が存在する」という主張である.
  - よって, この補題を満たすように方程式を変形すればよいとわかる.
- どのように変形すればよいのか?

方程式を増やすパート

- 係数が小さくなるように変形したいため, 定数倍操作に加えて「方程式同士を足し合わせ, 係数の大きさを調整する」という操作をする必要がある.
  - しかし, 求めたい解は同じままで変形をしたい.
  - そこで, 「同じ解を持つ方程式の和」を考える.

方程式を増やすパート

Proposition 8

M を法, $f(x) \equiv 0 \pmod{M}$, $g(x) \equiv 0 \pmod{M}$ を同じ解 x_0 を持つような *Modular* 方程式とする. このとき, 任意の整数 a, b について
$af(x) + bg(x) \equiv 0 \pmod{M}$ は x_0 を解に持つ.

- 証明は解 x_0 について

$$\begin{aligned} af(x_0) + bg(x_0) &\equiv a \cdot 0 + b \cdot 0 \\ &\equiv 0 \pmod{M} \end{aligned}$$

  より従う.
- 単純な命題だが, これは同じ解を持つ方程式同士の線形結合がまた同じ解を持つという重要な事実を示している.

方程式を増やすパート

- 係数の大きさを調整することが目的であるため, 暗黙のうちに線形独立な方程式であることが仮定されていることに注意
  - これらの条件を満たすような方程式を次ページで構築する.

方程式を増やすパート

- 与えられた $f(x) \equiv 0 \pmod{M}$ と同じ解を持つ方程式を構成する.
  - 同じ mod M の世界で構成することはできないが, M のべきの世界であれば構成可能.

Lemma 9

M を法, $f(x)$ を多項式とする. 自然数 m, ℓ について
$g_{i,j}(x) := M^{m-i}x^j f^i(x)$ $(0 \leq i \leq m, 0 \leq j \leq \ell)$ とおく. このとき,
$f(x_0) \equiv 0 \pmod{M}$ をみたす $x_0 \in \mathbb{Z}$ について, $g_{i,j}(x_0) \equiv 0 \pmod{M^m}$.

方程式を増やすパート

- 証明
  - 各 x_0 に対して k が存在して $f(x_0) = kM$ である. ゆえに $f^i(x_0) = k^i M^i$ と書くことができる.
  - $g_{i,j}(x)$ の定義より

$g_{i,j}(x_0) = M^{m-i} x_0^j f^i(x_0) = M^{m-i} x_0 k^i M^i = k^i x_0^j M^m$.

  - このことから, $g_{i,j}(x_0) \equiv 0 \pmod{M^m}$ が成立する (証明終).

方程式を増やすパート

- Lem. 9 によって, 複数の同じ解を持つ方程式を構成できた.
  - しかし, 係数については目的である Lem. 7 の条件と比較して極めて大きい.
- そこで, これら同じ解を持つ方程式の組み合わせによって Lem. 7 の条件を満たすものを構成したい.
  - そのために 格子 という道具を用いる. 次ページ以降, 格子の基礎を解説する.

方程式を組み合わせるパート - 格子

- 組み合わせパートにおいて格子という道具を使うため, ここで準備する.
  - 断りがない場合, ベクトルは縦ベクトルを表すこととする.

Definition 10 (格子)

ある自然数 n について $\mathbb{R}^n$ の基底 $\{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ をとる. このとき, それらの整数係数線型結合の全体の集合を $\mathbf{b}_1, \ldots, \mathbf{b}_n$ を基底とした格子といい, $\mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ で表す.

$$\mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_n) = \{ x_1\mathbf{b}_1 + \cdots + x_n\mathbf{b}_n \mid x_i \in \mathbb{Z} \}$$

- $\mathbf{b}_1, \ldots, \mathbf{b}_n$ はベクトルを並べた行列 $\mathbf{B} = {}^t(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ を用いて次のようにも表せる.

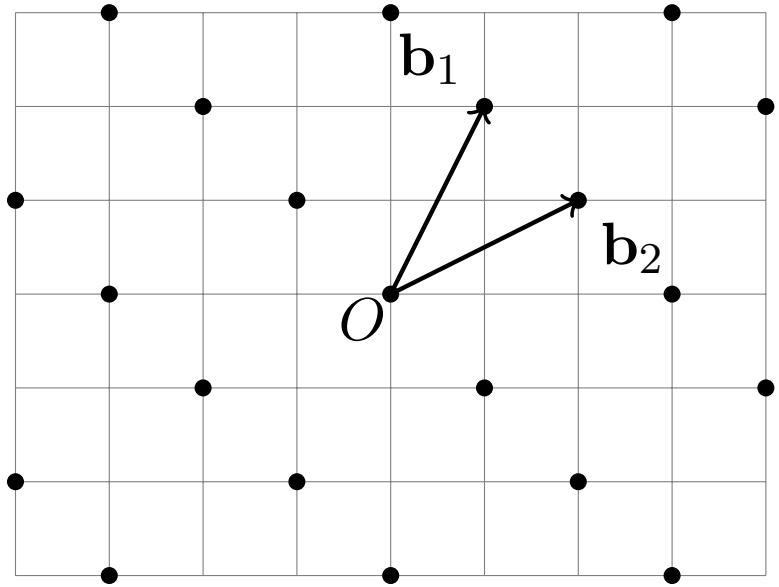
$$\mathcal{L}(B) = \{ \mathbf{Bx} \mid \mathbf{x} \in \mathbb{Z}^n \}$$

方程式を組み合わせるパート - 格子

- 用語の定義
  - $\mathbf{b}_1, \ldots, \mathbf{b}_n$: 基底ベクトル
  - $\mathbf{B} = {}^t(\mathbf{b}_1, \ldots, \mathbf{b}_n)$: 基底行列
    - ★ これらを合わせて格子基底と呼ぶ.
    - ★ 格子基底 $\mathbf{B}$ について $\mathcal{L}(\mathbf{B})$ を $\mathbf{B}$ が張る格子と呼ぶ.
  - $\mathcal{L}(\mathbf{B})$ の元: $(\mathcal{L}(\mathbf{B})$ の$)$ 格子点
  - $\mathcal{L}(\mathbf{B})$ の次数: n

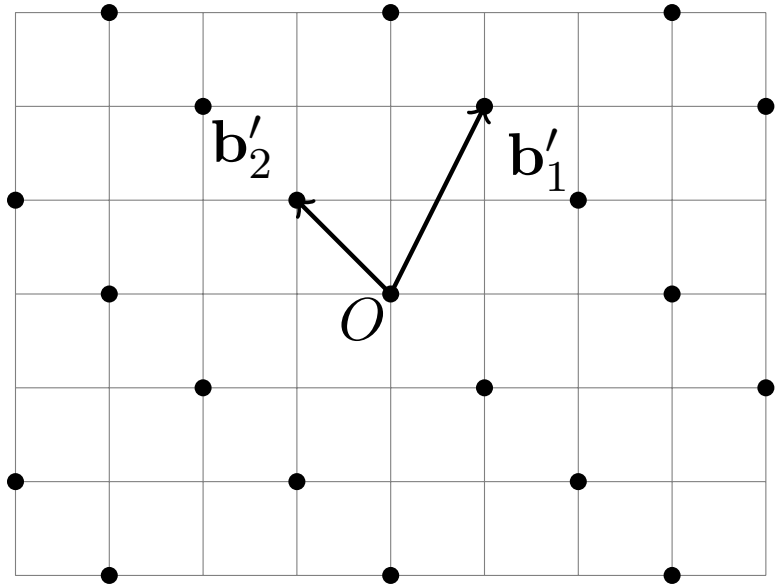
方程式を組み合わせるパート - 格子

- 例えば, $\mathbf{B} = \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}$ を基底行列とするとその格子は以下のように図示される.



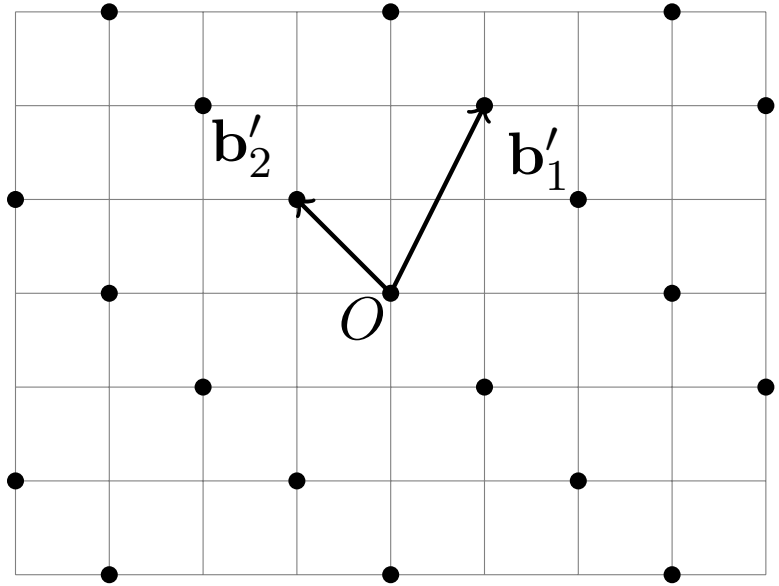
方程式を組み合わせるパート - 格子

- ここで, $\mathbf{B}' = \begin{pmatrix} 1 & 2 \\ -1 & 1 \end{pmatrix}$ という基底を持つ格子を同様に図示してみよう.



方程式を組み合わせるパート - 格子

- ここで, $\mathbf{B}' = \begin{pmatrix} 1 & 2 \\ -1 & 1 \end{pmatrix}$ という基底を持つ格子を同様に図示してみよう.



- 同じ格子が張られていることがわかる.

方程式を組み合わせるパート - 格子

- このように, 異なる基底 $\mathbf{B} \neq \mathbf{B}'$ について $\mathcal{L}(\mathbf{B}) = \mathcal{L}(\mathbf{B}')$ が成立することがある.
- 必要十分条件は以下の定理で示されている (証明略).

Theorem 11 ([Pei13] Lemma 2.5)

格子基底 $\mathbf{B}_1, \mathbf{B}_2$ が同じ格子を張ることは, $\mathbf{B}_1 = \mathbf{U}\mathbf{B}_2$ を満たすようなユニモジュラー行列 $\mathbf{U} \in \mathbb{Z}^{n \times n}$ が存在することに同値である. [a]

[a] ユニモジュラー行列とは, 行列式の絶対値が 1 であるような行列を指す.

方程式を組み合わせるパート - 格子

- Thm. 11 は, 基底行列を $\mathbf{B}$ としたとき, 任意のユニモジュラー行列 $\mathbf{U}$ について $\mathcal{L}(\mathbf{B}) = \mathcal{L}(\mathbf{UB})$ であることを言っている.
  - 先程の例 $\mathbf{B} = \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}$, $\mathbf{B}' = \begin{pmatrix} 1 & 2 \\ -1 & 1 \end{pmatrix}$ については,

  $\mathbf{U} = \begin{pmatrix} 1 & 0 \\ 1 & -1 \end{pmatrix}$ が該当する.
- また, ユニモジュラー行列の性質から $\mathcal{L}(\mathbf{B}) = \mathcal{L}(\mathbf{B}')$ ならば $|\det(\mathbf{B})| = |\det(\mathbf{B}')|$ もわかる.

方程式を組み合わせるパート - LLL

- 1982 年, Lenstra, Lenstra, and Lovász は [LLL82] においてこんにち **LLL** と呼ばれているアルゴリズムを提案した.
  - このアルゴリズムは格子基底を変換するものであり, 格子基底から同じ基底を張る より簡約化された 格子基底を入力される基底の大きさ [1] と次数の多項式時間で出力する.
  - 「簡約基底」については詳しく述べないが, イメージ的には「直交に近い又は直交した基底であり, 各ベクトルのノルムがより小さいもの」を表す.

[1] 各横ベクトルのノルムの最大値

方程式を組み合わせるパート - LLL

- 時間の都合上本スライドではアルゴリズムの詳細については踏み込まないため, アルゴリズムの出力する基底が満たす性質のみを Thm. 12 で示すのみとする.
  - 計算については, SageMath に既に関数が存在するため, これを利用する.

```
sage: M = Matrix(ZZ, [[30, 25], [58, 23]])
sage: M.LLL()

[ -2 -27]
[ 28  -2]
```

方程式を組み合わせるパート - LLL

Theorem 12 (LLL, [LLL82])

$\mathbf{B}$ を格子 $\mathcal{L}$ の基底の一つ, $\mathbf{B}' = {}^t(\mathbf{b}'_1, \mathbf{b}'_2, \ldots, \mathbf{b}'_n)$ を $\mathbf{B}$ を *LLL* アルゴリズムに入力した結果出力された基底とする. このとき, 以下が成立する:

$$\|\mathbf{b}'_1\| \leq \|\mathbf{b}'_2\| \leq \cdots \leq \|\mathbf{b}'_i\| \leq 2^{\frac{n(n-1)}{4(n+1-i)}} |\det(\mathbf{B})|^{\frac{1}{n+1-i}}.$$

特に, $\mathbf{b}'_1$ について

$$\|\mathbf{b}'_1\| \leq 2^{\frac{n-1}{4}} |\det(\mathbf{B})|^{\frac{1}{n}}$$

が成立する.

- この定理は, LLL アルゴリズムの出力が満たす条件を示している.

方程式を解くパート - アルゴリズム

- 同じ解を持つ方程式の生成, 格子, LLL と必要な基礎は揃った.
- ここからは, それらを組み合わせて実際に方程式を解くことのできるアルゴリズムを示す.
  - 最初にアルゴリズムを示し, その後に Step 毎の解説を入れる.

方程式を解くパート - アルゴリズム

- パラメータ: 整数 m, ℓ
  - m, ℓ の選び方について, 理論的考察がされている論文も存在するが, ここで紹介するアルゴリズムでは Heuristic な選び方をするものとする.
- 入力: 法 M, δ 次多項式 $f(x)$, 整数 X
  - $f(x)$ はモニックである必要がある.
- 出力: $|x_0| \leq X$, $f(x_0) \equiv 0 \pmod{M}$ を満たすような x_0, 又は失敗

方程式を解くパート - アルゴリズム

Step 1. $\bmod M^m$ で $f(x)$ と同じ根を持つ多項式 $g_{i,j}(x)$ $(0 \leq i \leq m, 0 \leq j \leq \ell)$ を計算する (Lem. 9).

Step 2. $g_{i,j}(x)$ の係数ベクトルから行列 B を構成する.

Step 3. LLL アルゴリズムを用いて格子基底 B から Thm. 12 を満たす格子基底 B' を計算する.

Step 4. B' の第 1 ベクトルの大きさを調べ, Howgrave-Graham's Lemma (Lem. 7) を満たすかチェックする.

Step 4.1. 条件を満たしていない場合は「失敗」を出力する.

Step 5. B' の第 1 ベクトルから多項式 $h(x)$ を構成する.

Step 6. $h(x) = 0$ の整数解を探し, その結果得られた解を出力する.

方程式を解くパート - 解説

- 具体的な Modular 方程式を交えてアルゴリズムを解説する.
- $M = 10007 \times 10009$ を法とし, 多項式 $f(x) = x^2 + 87814262x + 16117449$ について $f(x) \equiv 0 \pmod{M}$ の解を求める.
  - 実際の解は $x = 123, 12345678$ である.
- 解の最大値を $X = 200$ と設定し, $x = 123$ が求まることを確認することとする.

方程式を解くパート - 解説: Step 1/6

Step 1. $\bmod M^m$ で $f(x)$ と同じ根を持つ多項式 $g_{i,j}(x)$ $(0 \leq i \leq m,\ 0 \leq j \leq \ell)$ を計算する (Lem. 9).

- ここでは, m, ℓ はそれぞれ $m = 2$, $\ell = 1$ とする.
  - 全体の方程式の個数は $(m+1)(\ell+1) = 6$ 個.
- 計算自体は機械的に行うだけであるため, 計算結果を次ページに掲載した.

方程式を解くパート - 解説: Step 1/6

$$
\begin{aligned}
g_{0,0}(x) &= 1003203822016396 9 \\
g_{0,1}(x) &= 1003203822016396 9x \\
g_{1,0}(x) &= 100160063x^2 + 8795482014218506x + 1614324707239287 \\
g_{1,1}(x) &= 100160063x^3 + 8795482014218506x^2 + 1614324707239287x \\
g_{2,0}(x) &= x^4 + 175628524x^3 + 7711344642839542x^2 \\
&\quad + 2830683778515276x + 259772162267601 \\
g_{2,1}(x) &= x^5 + 175628524x^4 + 7711344642839542x^3 \\
&\quad + 2830683778515276x^2 + 259772162267601x
\end{aligned}
$$

方程式を解くパート - 解説: Step 2/6

Step 2. $g_{i,j}(x)$ の係数ベクトルから行列 B を構成する.

- Howgrave-Graham's Lemma のため, 実際の係数ベクトルの計算には $g_{i,j}(x)$ ではなく $g_{i,j}(xX)$ を用いることに注意しつつ, 係数ベクトルから行列を作る.
- 係数ベクトルから行列を作る操作については, 連立方程式の行列解法において $ax + by = s$, $cx + dy = t$ という 2 本の方程式から

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} s \\ t \end{pmatrix}$$

という行列方程式を作る操作を拡張したものだと考えればよい.

方程式を解くパート - 解説: Step 2/6

- $g_{i,j}(xX)$ の係数ベクトルから行列を作る手順を具体的に示す.
  - $g_{i,j}(x)$ の k 次の係数のことを $g_{i,j}^{(k)}$ と表すとする.
  - このとき, 係数ベクトルから作られた行列は $(\delta+1)(m+1)$ 次の正方行列であり, 以下のような形を持っている.

	1	x	x^2	x^3	x^4	x^5
$g_{0,0}$	$g_{0,0}^{(0)}$	$g_{0,0}^{(1)}X$	$g_{0,0}^{(2)}X^2$	$g_{0,0}^{(3)}X^3$	$g_{0,0}^{(4)}X^4$	$g_{0,0}^{(5)}X^5$
$g_{0,1}$	$g_{0,1}^{(0)}$	$g_{0,1}^{(1)}X$	$g_{0,1}^{(2)}X^2$	$g_{0,1}^{(3)}X^3$	$g_{0,1}^{(4)}X^4$	$g_{0,1}^{(5)}X^5$
$g_{1,0}$	$g_{1,0}^{(0)}$	$g_{1,0}^{(1)}X$	$g_{1,0}^{(2)}X^2$	$g_{1,0}^{(3)}X^3$	$g_{1,0}^{(4)}X^4$	$g_{1,0}^{(5)}X^5$
$g_{1,1}$	$g_{1,1}^{(0)}$	$g_{1,1}^{(1)}X$	$g_{1,1}^{(2)}X^2$	$g_{1,1}^{(3)}X^3$	$g_{1,1}^{(4)}X^4$	$g_{1,1}^{(5)}X^5$
$g_{2,0}$	$g_{2,0}^{(0)}$	$g_{2,0}^{(1)}X$	$g_{2,0}^{(2)}X^2$	$g_{2,0}^{(3)}X^3$	$g_{2,0}^{(4)}X^4$	$g_{2,0}^{(5)}X^5$
$g_{2,1}$	$g_{2,1}^{(0)}$	$g_{2,1}^{(1)}X$	$g_{2,1}^{(2)}X^2$	$g_{2,1}^{(3)}X^3$	$g_{2,1}^{(4)}X^4$	$g_{2,1}^{(5)}X^5$

方程式を解くパート - 解説: Step 2/6

- 実際には, この行列は下三角行列である.
  - また, 元の多項式 $f(x)$ がモニックであることから最高次係数は $M^{m-i}x^j$ である.
- これを踏まえて, 実際の行列を次ページに示す.
  - ただし, 行列のうちこの後必要になる対角成分以外は省略する.

方程式を解くパート - 解説: Step 2/6

$$B = \begin{pmatrix} M^2 & & & & \\ & M^2X & & & \\ - & - & MX^2 & & \\ & - & - & MX^3 & \\ - & - & - & - & X^4 \\ & - & - & - & - & X^5 \end{pmatrix}$$

マイナス記号は数値, 空白は 0 を表している.

方程式を解くパート - 解説: Step 3/6

Step 3. LLL アルゴリズムを用いて格子基底 B から Thm. 12 を満たす格子基底 B' を計算する.

- 行列式は対角成分の積として $\det(B) = M^{2+2+1+1}X^{0+1+2+3+4+5} = M^6X^{15}$ と計算できる.
- この基底について, LLL アルゴリズムを適用した後の基底を $B' = {}^t(\mathbf{b}'_1, \ldots, \mathbf{b}'_6)$ とする.

方程式を解くパート - 解説: Step 4/6

Step 4. B' の第 1 ベクトルの大きさを調べ, Howgrave-Graham's Lemma (Lem. 7) を満たすかチェックする.

- Thm. 11 より, 異なる基底行列 B, B' についても, それぞれ同じ格子を張るならば, ユニモジュラ行列 U が存在して $B' = UB$ と表せる.
  - よって, 第 1 ベクトル $\mathbf{b}'_1$ は行列 B のベクトル $\mathbf{b}_1, \ldots, \mathbf{b}_6$ の線型結合で表される.
- Prop. 8 より, $\mathbf{b}'_1$ を係数ベクトルとする多項式 $h(x)$ は $g_{i,j}(x)$ と同じ解を持つ.
  - 加えて, Thm. 12 より, このベクトルの大きさは必ずある程度小さいことが保証されている.
    - ゆえに, LLL を利用することで Howgrave-Graham's Lemma を満たすような多項式を作っているのだと言える.
- 実際にこれを確かめる.

方程式を解くパート - 解説: Step 4/6

- Thm. 12 より, 第 1 ベクトル $\mathbf{b}'_1$ のノルムについて

$$\|\mathbf{b}'_1\| \leq 2^{\frac{6-1}{4}}(M^6X^{15})^{\frac{1}{6}} \tag{1}$$

  という不等式が成立する (事実).
- また, 実際の B' の第 1 ベクトル $\mathbf{b}'_1$ の単項式の数は 6 であるため, Howgrave-Graham's Lemma より

$$\|\mathbf{b}'_1\| \leq \frac{M^2}{\sqrt{6}} \tag{2}$$

  が成立していれば $\mathbf{b}'_1$ を係数ベクトルとした方程式は整数方程式として解くことができると結論付けられる (成り立っていてほしい条件).

方程式を解くパート - 解説: Step 4/6

- Thm. 12 により得られた不等式 Eq. 1 と Howgrave-Graham's Lemma により得られた不等式 Eq. 2 を組み合わせて考えると,

$$2^{\frac{6-1}{4}}(M^6X^{15})^{\frac{1}{6}} \leq \frac{M^2}{\sqrt{6}} \tag{3}$$

が得られる. これを具体的に計算すると

$$2^{\frac{6-1}{4}}(M^6X^{15})^{\frac{1}{6}} = 1347587806852 57.9353824219819...$$

$$\frac{M^2}{\sqrt{6}} = 4095562453250075.372969413645...$$

であり, Eq. 3 が成立していることがわかる.

方程式を解くパート - 解説: Step 5/6

Step 5. B' の第 1 ベクトルから多項式 $h(x)$ を構成する.

- $\mathbf{b}_1'$ の i 番目の成分が x^i の係数となるようにすればよい.
  - 実際には $g_{i,j}(xX)$ の係数ベクトルから格子を構成したことを踏まえて, $\mathbf{b}_1'$ の i 番目の成分を X^i で割ったものを x^i の係数とする.
  - 実際に得られた $\mathbf{b}_1'$ から構成した多項式は以下のとおりである.

$$h(x) = -6x^5 - 5972x^4 - 713119x^3 + 178357141x^2 + 31295860138x - 3684907007880$$

方程式を解くパート - 解説: Step 6/6

Step 6. $h(x) = 0$ の整数解を探し，その結果得られた解を出力する．

- 先ほどみたように，この方程式は Howgrave-Graham's Lemma を満たしていることがわかっているため，整数方程式として解くことで解 $x = 123$ を得ることができることが確定している．
  - Thm. 12 の条件は満たさなくとも，実際に得られる第 1 ベクトルのノルムは満たしている場合もあるため，とりあえず LLL にかけてみるという方針はひとつ戦略としてあり．
- 以下は SageMath によってこれを解いた結果である．

```
sage: PR.<x> = PolynomialRing(ZZ)
sage: f = -6*x^5 - 5972*x^4 - 713119*x^3 + 178357141*x^2 + 31295860138*x - 3684907007888
sage: f.roots()
[(123, 1)]
sage:
```

方程式を解くパート - 解説

- 実例を交えて Coppersmith's Method [HG97] を解説した.
- SageMath を利用してこれを実装したものが以下である.

`https://gist.github.com/elliptic-shiho/8574910f28c8fdbbd01b8193aa4af6af`

方程式を解くパート - 解説

- これにより, 任意の 1 次 Modular 方程式について, 解の最大値 X によってはそれを解くことができるようになった.
  - これで Thm. 4 のアルゴリズムを構成したことになる.
- 次に, Thm. 6 についても示す.
  - 未知の法 b について, その倍数 M が既知ならば解けると主張する定理
  - 重要なのは補題一つのみである.

Coppersmith's Method - Thm.6

- 与えられた $f(x) \equiv 0 \pmod{b}$ と同じ解を持つ方程式を構成する.
  - b は未知だが, その倍数 M は既知であるとする.
  - 同じ mod b で構成することはできないが, b のべきであれば構成可能.

Lemma 13

b を未知の法, M を既知の整数とし, b は M を割り切るとする. $f(x)$ を多項式とし, 自然数 m, ℓ について $g_{i,j}(x) := M^{m-i}x^j f^i(x)$ $(0 \leq i \leq m,$ $0 \leq j \leq \ell)$ とおく. このとき, $f(x_0) \equiv 0 \pmod{b}$ をみたす $x_0 \in \mathbb{Z}$ について, $g_{i,j}(x_0) \equiv 0 \pmod{b^m}$.

Coppersmith's Method - Thm.6

- 証明
  - $t = \frac{M}{b}$ とおく.
  - 各 x_0 に対して k が存在して $f(x_0) = kb$ である. ゆえに $f^i(x_0) = k^i b^i$ と書くことができる.
  - $g_{i,j}(x)$ の定義より $g_{i,j}(x_0) = M^{m-i} x_0^j f^i(x_0) = (b \times t)^{m-i} x_0 k^i b^i = t^{m-i} k^i x_0^j b^m$.
  - このことから, $g_{i,j}(x_0) \equiv 0 \pmod{b^m}$ が成立する (証明終).

Coppersmith's Method - Thm.6

- Lem. 13 により, $\mathrm{mod}\ b^m$ で同じ解を持つ複数の方程式を構成できた.
  - これらを先程のアルゴリズムのように LLL に通し, Howgrave-Graham's Lemma のチェックを介せば同様に解くことができる.
  - よって, Thm. 6 のアルゴリズムも構成できたといえる.

Coppersmith's Method - 多変数の場合

- 1 変数の場合に示せたのであれば, 2 変数や 3 変数, もっと言えば n 変数についても示せそうに思える.
  - 2 変数の場合について少し考察する.
- 一般の n 変数の場合についても拡張することは容易である.

Coppersmith's Method - 多変数の場合

- まず, Howgrave-Graham's Lemma の 2 変数版のステートメントを示す.

Lemma 14 (Howgrave-Graham for Bivariate Polynomial)

M を法, $g(x,y) \in \mathbb{Z}[x,y]$ を整数多項式とし, 含まれる単項式の数を ω とする. $g(x,y)$ に対してある X, Y が存在し, $g(x_0,y_0) \equiv 0 \pmod{M}$ なる $x_0, y_0 \in \mathbb{Z}$ について $|x_0| \leq X$, $|y_0| \leq Y$ であると仮定する. このとき,

$$\|g(xX, yY)\| < \frac{M}{\sqrt{\omega}}$$

が成立するならば $g(x_0,y_0) = 0$ が (整数方程式として) 成立する.

Coppersmith's Method - 多変数の場合

- 証明は実は 1 変数のときとほぼ変わらない.
  - 時間の都合上スライドでは省略
- 同様に, Howgrave-Graham's Lemma については n 変数への拡張も容易.

Coppersmith's Method - 多変数の場合

- 次に, 方程式を複数生成するパートの拡張について考察する.

Lemma 15

b を未知の法, M を既知の整数とし, b は M を割り切るとする. $f(x,y)$ を多項式とし, 自然数 m, ℓ, τ について $g_{i,j,k}(x,y) := M^{m-i}x^j y^k f^i(x,y)$ $(0 \leq i \leq m,\ 0 \leq j \leq \ell,\ 0 \leq k \leq \tau)$ とおく. このとき, $f(x_0,y_0) \equiv 0 \pmod{b}$ をみたす $x_0, y_0 \in \mathbb{Z}$ について, $g_{i,j,k}(x_0,y_0) \equiv 0 \pmod{b^m}$.

- 証明は略するが, 示す方針, 内容はほぼ同じである.

Coppersmith's Method - 多変数の場合

- LLL に使う基底行列への係数の並べ方は, 全ベクトル内で一定でありさえすればどのような順序でもよいことに注意すれば, これらの道具を使えば既に 2 変数に対応した Coppersmith's Method は示せていることに気づくだろう.
  - n 変数になると Heuristic に単項式のとり方を改良した手法が提案されている [JM06] が, 基本的な考え方は今まで見てきたものの応用である.

Coppersmith's Method - 多変数の場合

- Q. 2 変数の場合, 整数方程式に変換できたとしても解けないのでは?
  - A. 消去式 (resultant) や Gröbner 基底を用いる.
- Gröbner 基底については後で述べる. 消去式とは, 簡単に言えば 2 変数多項式 $f(x, y)$, $g(x, y)$ がそれぞれ線形独立, かつ共通根 x_0, y_0 を持つとき, y についての消去式 $h = \mathrm{Res}_y(f, g)$ により得られる h は 1 変数多項式であり, $h(x_0) = 0$ を満たす.
  - 解説した手法では LLL の条件から得られるベクトルの長さの条件は第 1 ベクトルしか見ていなかったが, 2 変数の場合は先頭 2 つのベクトルについて条件を見なければならないことがわかる.
    - ⋆ このように, 一般に変数の数が増えれば増えるほど解くための条件が厳しくなるため, Coppersmith's Method はなるべく変数の数を減らす方向で考察する必要がある.

Coppersmith's Method - 多変数の場合

- Q. β はどこにいったのか
  - A. β は Howgrave-Graham の条件において利用しているだけなので, 本スライドにおいてはあえて触れていない
    - ★ 考え方としては Eq. 3 の条件式をより一般に考察すると得られる.
  - 詳しい話は [May03].

Unravalled Linerization

- Coppersmith's Method のアルゴリズムでは, Thm. 12 の条件式を見てわかるとおり入力する基底行列の行列式が性能に大きく影響する.
  - このため, 項の数を減らす, 次数を下げる, 解の最大値に制約を設ける等のいくつかの試行錯誤がなされてきた.
- 実は, 2009 年に行列式の大きさを大幅に小さく抑えることができる手法が [HM09] により提案された.

Unravalled Linerization

- 例として Boneh-Durfee's Attack でこの手法を解説する.
  - 最初の方で見ていたように, この攻撃手法では以下の式を解くことが目的だった.

$$x(A + y) + 1 \equiv 0 \pmod{e}.$$

Unravalled Linerization

- 多項式 $f(x,y) = xy + xA + 1$ と $\bmod e^m$ で同じ根を持つ多項式を他に作ることを考えたとき, xy の項が実は邪魔になる.
  - y の大きさが少し大きいため, 今まで見てきたとおり $x^i y^j e^{m-k} f^k(x)$ の形でこれを構成しようとすると, 行列式が極端に大きくなってしまう.
    - ⋆ 行列式に落としたときに根の最大値 X, Y について対角成分が $X^i Y^j e^{m-k}(XY)^k$ の形で表されることを考えると, X, Y の大きさと i, j によっては $e^m \leq X^i Y^j e^{m-k}(XY)^k$ となってしまうことがある.
    - ⋆ これは行列式を小さくする際に大きなボトルネックになるだけでなく, 結果的に係数の調整に用いるためにも大きすぎるため用いられないという二重の問題を抱えている厄介な多項式になってしまっている.

Unravalled Linerization

- 実は, より一般的な定義では格子は非正方行列を基底にとることができる.
  - Boneh-Durfee[BD99] ではこの一般的な定義の方を採用し, そのような厄介な多項式を含む行を取り除いた非正方行列による格子を考察した.
  - 行列式を用いずに格子の大きさを測る別の手法に加え, 独自手法を編み出してその大きさを基底行列から見積もり, 結果的に $d < n^{0.292}$ を達成した.
  - しかし, あまりに複雑過ぎて利用されることは少なく, その手法を用いた考察もさほど普及しなかった.

Unravalled Linerization

- Herrmann, May が [HM09] で提案した **Unravelled Linearization** という手法では, この問題を初等的な変形のみで解決している.
  - まず, 非線形項をまとめる変数変換 $u = xy + 1$ を用いて式を次のように変換する.

$$\underbrace{xy + 1}_{u} + Ax \equiv 0 \pmod{e}$$

$$\Downarrow$$

$$u + Ax \equiv 0 \pmod{e}$$

Unravalled Linerization

- 変数変換によって得られた式 $f(u, x) = u + Ax$ について, 少し変則的な変数変換の方法を考える.
  - 具体的には, 方程式を増やす時に用いる多項式を

$$g_{i,j,k}(u, x, y) = x^i y^j e^{m-k} f^k(u, x)$$

とする.

  - u については増やさず, x と y のみに注目していることがわかる.

Unravalled Linerization

- まず, 目的としては係数の調整が目的である.
  - ゆえに, x のみに注目したり y のみに注目することは性能劣化につながる.
  - しかし, 今までの構成手法では x, y 両方に注目すると非効率的な (unhelpful な) 多項式ができてしまった.
- そこで, あまり係数について注目する必要のない非線形項 $xy + 1$ については変数 u にまとめてしまい, それ以外の x, y についての項は通常通り係数調整が可能なようにする.
  - こうすることで, 行列式には影響を及ぼさず, 方程式の形を崩すこともなく, それでいて行列式の値を小さく抑えることが可能となる.

Unravalled Linerization

- LLL を通した後に再度 $u = xy + 1$ の関係式を用いて逆変換することで x, y についての式に戻すことができる.
  - このような「u による注目したくない項の線形化 + LLL 後の逆変換」という手法のことを Unravelled Linearization と呼ぶ.
  - 実際, この手法によって Boneh-Durfee's Attack に用いられる格子についての議論はより簡略化され, 代数的な変形によって全ての証明が完了するようになった.
  - この手法については提案論文である [HM09] のほか, Herrmann の博士論文 [Her11], Boneh-Durfee's Attack への適用論文 [HM10] も参考になる.

0CTF 2017 Finals: Authentication&Secrecy

0CTF 2017 Finals: Authentication&Secrecy

- この問題は, RSA の公開鍵, 暗号文と鍵生成アルゴリズムが与えられる典型的な解読問題.
- 配布ファイルは以下の通り
  - `alice_public_key.py`
  - `bob_public.key.py`
  - `communication.py`
  - `keygen.sage`
  - `send.bak`

0CTF 2017 Finals: Authentication&Secrecy

- Alice, Bob の公開鍵を見てみると, それぞれ (e, N_1, N_2) の組が入っていることがわかる.
  - また, e がそこそこ大きい値になっている. このことから, d が小さいときに利用できた Boneh-Durfee のような Small Secret Exponent Attack を疑う.
- 確かめるために, `keygen.sage` を読む.

0CTF 2017 Finals: Authentication&Secrecy

- 鍵生成は配布された `keygen.sage` の以下の部分で行われている

```
def genKey(n, nd):
    '''
    Designed with the awesome idea that generating two RSA key pairs
    that have the same public and private exponents to reduce the
    storage requirements!
    '''

    assert n/2 > nd
    tmp = n/2 - nd

    while True:
        while True:
            x1 = randrange(2^(nd-1), 2^nd)
            x2 = randrange(2^(tmp-1), 2^tmp)
            p1 = x1*x2 + 1
            if p1.is_prime():
                print '[+]bit length of p1:', int(p1).bit_length()
                break

        while True:
            y2 = randrange(2^(tmp-1), 2^tmp)
            p2 = x1*y2 + 1
            if p2.is_prime():
                print '[+]bit length of p2:', int(p2).bit_length()
                break

        while True:
            y1 = randrange(2^(nd-1), 2^nd)
            q1 = y1*y2 + 1
            if q1.is_prime():
                print '[+]bit length of q1:', int(q1).bit_length()
                break
```

```
        count = 0
        while True:
            d = randrange(2^(nd-1), 2^nd)
            if gcd(x1*x2*y1*y2, d) != 1:
                continue
            e = int(1/Mod(d, (p1-1)*(q1-1)))
            k1 = (e*d - 1) // ((p1-1)*(q1-1))
            assert e*d == (p1-1)*(q1-1)*k1 + 1
            q2 = k1*x2 + 1
            if q2.is_prime():
                print '[+]bit length of q2:', int(q2).bit_length()
                break
        n1 = p1 * q1
        n2 = p2 * q2
        n1, n2 = max([n1, n2]), min([n1, n2])
        if n1 <= threshold:
            print '[+]N for encryptions is too small!'
            continue
        if n2 >= threshold:
            print '[+]N for signatures is too large!'
            continue
        break

    assert int(pow(pow(0xdeadbeef, e, n1), d)) == 0xdeadbeef
    assert int(pow(pow(0xdeadbeef, e, n2), d)) == 0xdeadbeef
    return (e, d, n1, n2)
```

0CTF 2017 Finals: Authentication&Secrecy

```
def genKey(n, nd):
    '''
    Designed with the awesome idea that generating two RSA key pairs
    that have the same public and private exponents to reduce the
    storage requirements!
    '''

    assert n/2 > nd
    tmp = n/2 - nd
```

- 引数は n, n_d.
  - n_d の大きさが n の半分以下という条件が気になる.
- 変数 tmp は $tmp = n/2 - n_d$ と定義している.

0CTF 2017 Finals: Authentication&Secrecy

```
while True:
    x1 = randrange(2^(nd-1), 2^nd)
    x2 = randrange(2^(tmp-1), 2^tmp)
    p1 = x1*x2 + 1
    if p1.is_prime():
        print '[+]bit length of p1:', int(p1).bit_length()
        break
```

- まず, 乱数 x_1, x_2 をそれぞれ n_d, tmp ビットの大きさを持ち, かつ $p_1 = x_1 x_2 + 1$ が素数となるように生成する.
  - すなわち, $p_1 = x_1 x_2 + 1$ のビット長は大体 $n/2$ 程度の大きさである.

0CTF 2017 Finals: Authentication&Secrecy

```
while True:
    y2 = randrange(2^(tmp-1), 2^tmp)
    p2 = x1*y2 + 1
    if p2.is_prime():
        print '[+]bit length of p2:', int(p2).bit_length()
        break
```

- 次に, y_2 を tmp ビットの大きさを持ち, $p_2 = x_1 y_2 + 1$ が素数となるように生成する.
  - p_2 のビット長もまた $n/2$ 程度の大きさである.

0CTF 2017 Finals: Authentication&Secrecy

```
while True:
    y1 = randrange(2^(nd-1), 2^nd)
    q1 = y1*y2 + 1
    if q1.is_prime():
        print '[+]bit length of q1:', int(q1).bit_length()
        break
```

- y_1 がビット長 n_d, かつ $q_1 = y_1 y_2 + 1$ が素数となるように生成する.
  - q_1 の大きさも $n/2$ 程度といえる.

0CTF 2017 Finals: Authentication&Secrecy

```
count = 0
while True:
    d = randrange(2^(nd-1), 2^nd)
    if gcd(x1*x2*y1*y2, d) != 1:
        continue
    e = int(1/Mod(d, (p1-1)*(q1-1)))
    k1 = (e*d - 1) // ((p1-1)*(q1-1))
    assert e*d == (p1-1)*(q1-1)*k1 + 1
    q2 = k1*x2 + 1
    if q2.is_prime():
        print '[+]bit length of q2:', int(q2).bit_length()
        break
```

- この部分を解説する前に, 一旦ここまでで生成した値を整理する.
  - x_1, x_2, y_1, y_2, $p_1 = x_1x_2 + 1$, $p_2 = x_1y_2 + 1$, $q_1 = y_1y_2 + 1$

0CTF 2017 Finals: Authentication&Secrecy

- まず, p_1, q_1, p_2 はそれぞれ素数, かつ $n/2$ の大きさを持つ.
  - 実際に生成される値は n-bit RSA の法であることを考えると自然.
- $N_1 = p_1 q_1$ とすると, $\phi(N_1) = (p_1 - 1)(q_1 - 1) = x_1 x_2 y_1 y_2$ である.
  - これを踏まえてコードをもう一度読む.

0CTF 2017 Finals: Authentication&Secrecy

```
count = 0
while True:
    d = randrange(2^(nd-1), 2^nd)
    if gcd(x1*x2*y1*y2, d) != 1:
        continue
    e = int(1/Mod(d, (p1-1)*(q1-1)))
    k1 = (e*d - 1) // ((p1-1)*(q1-1))
    assert e*d == (p1-1)*(q1-1)*k1 + 1
    q2 = k1*x2 + 1
    if q2.is_prime():
        print '[+]bit length of q2:', int(q2).bit_length()
        break
```

- 読む限り, ここでは d, q_2 を生成しているのだとわかる.
  - d と $x_1x_2y_1y_2$ が互いに素であることをチェックしているのは, 先ほど見たとおり $\phi(N_1)$ と互いに素であることのチェックに等しい.

0CTF 2017 Finals: Authentication&Secrecy

```
count = 0
while True:
    d = randrange(2^(nd-1), 2^nd)
    if gcd(x1*x2*y1*y2, d) != 1:
        continue
    e = int(1/Mod(d, (p1-1)*(q1-1)))
    k1 = (e*d - 1) // ((p1-1)*(q1-1))
    assert e*d == (p1-1)*(q1-1)*k1 + 1
    q2 = k1*x2 + 1
    if q2.is_prime():
        print '[+]bit length of q2:', int(q2).bit_length()
        break
```

- 次に, $e = d^{-1} \mod (p_1 - 1)(q_1 - 1)$ で e を生成し, $\frac{ed-1}{(p_1-1)(q_1-1)}$ を計算している.
  - $ed \equiv 1 \pmod{\phi(N_1)}$ について, ある k_1 が存在し, $ed = 1 + k_1\phi(N_1)$ と書くことができることから, これは k_1 のみを取り出しているということになる.

0CTF 2017 Finals: Authentication&Secrecy

```
count = 0
while True:
    d = randrange(2^(nd-1), 2^nd)
    if gcd(x1*x2*y1*y2, d) != 1:
        continue
    e = int(1/Mod(d, (p1-1)*(q1-1)))
    k1 = (e*d - 1) // ((p1-1)*(q1-1))
    assert e*d == (p1-1)*(q1-1)*k1 + 1
    q2 = k1*x2 + 1
    if q2.is_prime():
        print '[+]bit length of q2:', int(q2).bit_length()
        break
```

- その後, $q_2 = x_2 k_1 + 1$ とおき, この q_2 が素数であることをチェックしている.
  - k_1 と d の大きさはほぼ同じになるため, q_2 の大きさは $n/2$ 程度の大きさになる.

0CTF 2017 Finals: Authentication&Secrecy

- $N_2 = p_2 q_2$ とおくと, $\phi(N_2) = (p_2 - 1)(q_2 - 1) = x_1 x_2 k_1 y_2$ である.
  - $ed = 1 + k_1(x_1 x_2 y_1 y_2)$ であったことを踏まえて式変形すると,

$$\begin{aligned} ed &= 1 + k_1 \phi(N_1) \\ &= 1 + k_1(x_1 x_2 y_1 y_2) \\ &= 1 + y_1(x_1 x_2 k_1 y_2) \\ &= 1 + y_1 \phi(N_2) \end{aligned}$$

- よって, このように生成した N_1, N_2 は同じ e, d を共有する RSA 鍵の法である.

0CTF 2017 Finals: Authentication&Secrecy

- ここまでをまとめると, 次の定義になる.

Definition 16

素数 p_1, q_1, p_2, q_2 について, $N_1 = p_1q_1$, $N_2 = p_2q_2$ とおく. e, d を $ed \equiv 1 \pmod{\phi(N_1)}$, $ed \equiv 1 \pmod{\phi(N_2)}$ を満たすようにとり, $(e, N_1), (e, N_2)$ をそれぞれ RSA 公開鍵, d を共通の RSA 秘密鍵とする.

- この定義を手がかりに, この RSA Variant について調べる.

0CTF 2017 Finals: Authentication&Secrecy

- RSA の Variant についての問題は, 多くの場合問題名や途中の文言, 似た Variant について掲載されている文献を読み漁るとよい.
  - この Variant も含め複数の Variant を紹介している文献としては [Hin07] がおすすめ.
- この問題の場合, 「Authentication & Secrecy」という文言で検索すると数々の論文がヒットする **Dual RSA** がその Variant であった.

0CTF 2017 Finals: Authentication&Secrecy

- さて, 鍵生成コードの genkey 関数は第 1 引数で N_1, N_2 のビットサイズ, 第二引数で d のビットサイズを指定できた.
  - この問題ではそれぞれ 1024, 342 となっている. ここから d の大きさを概算すると $\frac{342}{1024} = 0.33398...$ となる.
  - 言い換えると, d は $N = \max(N_1, N_2)$ とおいても $N^{0.334}$ 未満の大きさしか持っていない.
- しかし, d が小さい場合に適用可能な手法のうち最も良い結果を出している Boneh-Durfee's Attack は $d < N^{0.292}$ の場合にしか適用できない.

0CTF 2017 Finals: Authentication&Secrecy

- 往々にして, RSA の Variant は RSA よりも複雑な構造を持っていることが多い.
  - そして, 攻撃に応用する際にその構造を利用することができる. ゆえに, Variant の場合は RSA よりもゆるい条件の元で同じ問題が解けることがある.
- Dual RSA についても同様で, d が $N^{0.334}$ 程度の大きさであったとしても解くことができる手法 [PHL+17] が存在した.
  - 提案は 2014 年, 出版はなんと 2017 年.
  - 定理の主張を次ページに示す.

0CTF 2017 Finals: Authentication&Secrecy

Theorem 17 ([PHL+17])

(e, N_1), (e, N_2) を *Dual RSA* の公開鍵, d を秘密鍵とする. このとき, $d < N^{\frac{9-\sqrt{21}}{12}+\epsilon}$ の条件の元で N_1, N_2 を効率よく素因数分解することができる. ただし, ϵ は誤差項.

- $\frac{9-\sqrt{21}}{12} = 0.368...$ であることから, この攻撃手法を用いることで問題は解くことができる.

0CTF 2017 Finals: Authentication&Secrecy

- 手法を概説する.
  - まず, $N_1 = p_1q_1$, $N_2 = p_2q_2$ について k_1, k_2 が存在して次の 2 式が成立する.

$$ed = 1 + k_1(N_1 - (p_1 + q_1 - 1))$$
$$ed = 1 + k_2(N_2 - (p_2 + q_2 - 1))$$

  - まず, 上の N_1 についての式に注目し, この d を表す式をより簡単に表現する方法を考える.

0CTF 2017 Finals: Authentication&Secrecy

- これには LLL を用いる.
- ある大きい整数 A について, 行列

$$M = \begin{pmatrix} A & 0 \\ e & N_1 \end{pmatrix}$$

の張る格子 $\mathcal{L}(M)$ を考える.

- $v = {}^t(d, -k_1)$ とおくと,

$$Mv = \begin{pmatrix} Ad \\ ed - k_1 N_1 \end{pmatrix}$$

である. すなわち, $t = {}^t(Ad, ed - k_1 N_1)$ は $\mathcal{L}(M)$ の格子点である.

  - このことは, t というベクトルは $\mathcal{L}(M)$ の基底の線型結合で表せるということを表している.

0CTF 2017 Finals: Authentication&Secrecy

- 勉強会中にあった質問により追記: この格子基底が何故選ばれたのかについて解説する [2].
  - まず, 目的そのものは「d についての別表現をもとめること」である.
  - その例としては, 例えば a_1, x_1, a_2, x_2 という 4 つの整数 (x_1, x_2 が既知という仮定) があり $d = a_1x_1 + a_2x_2$ と表せるような分解がある.
- この別表現はそこまで難しい問題ではない. 例えば $x_1 = 1$, $x_2 = 0$, $a_1 = d$, $a_2 = 0$ という指定をすればよい.
  - しかし, ここまで自明な問題は解きづらい. よって, このような x_1, x_2 で非自明なものを求める問題を考える.

[2] [PHL+17] 中では一切触れられていない.

0CTF 2017 Finals: Authentication&Secrecy

- 先の式は整数線型結合とみなせる. よって, 格子点の成分で d が含まれるものがあればよいのでは, という発想ができる.
  - d と関連があり, かつ線形独立な値 $ed - k_1 N_1 = -k_1(p_1 + q_1 - 1)$ が含まれるような格子として $B = \begin{pmatrix} 1 & 0 \\ e & N_1 \end{pmatrix}$ を基底に持つようなものを考える. この格子 $\mathcal{L}(B)$ については

$$B \begin{pmatrix} d \\ -k_1 \end{pmatrix} = \begin{pmatrix} d \\ ed - k_1 N_1 \end{pmatrix}$$

であり, d が成分として含まれるような基底が構成できていることがわかる.

0CTF 2017 Finals: Authentication&Secrecy

- この基底を LLL により変換した基底が非自明なものとなっていれば, 当初の目的は達成できる.
  - しかし, この基底にはひとつ問題が存在する.
- $(1, 0)$ というベクトルのノルムは他のベクトルに比べて非常に小さい.
  - そのため, LLL を通したとしてもこのベクトルが最短となり, 求めたい x_1, x_2 が求まらない可能性がある.
- そこで, 十分大きな整数 A を用いてベクトル $(A, 0)$ のノルムが LLL 条件よりも十分大きくなるようにし, LLL 後の格子を操作した.
  - この操作により得られた基底は $\begin{pmatrix} A & 0 \\ e & N_1 \end{pmatrix}$ である. これはこの手法で用いている基底に等しい.
  - このように, 整数による重みつけでノルムを操作し, LLL 後の格子に含まれる点を調整する手法を decoration(デコレーション) という.

0CTF 2017 Finals: Authentication&Secrecy

- M を LLL により変換した基底を $M' = \begin{pmatrix} \lambda_1 \\ \lambda_2 \end{pmatrix}$ とする.
  - すると, ある $a_1, a_2 \in \mathbb{Z}$ があって $t = a_1\lambda_1 + a_2\lambda_2$ である.
  - $\lambda_1 = (l_{11}, l_{12})$, $\lambda_2 = (l_{21}, l_{22})$ とすると $Ad = a_1 l_{11} + a_2 l_{21}$.
  - 両辺を A で割れば $d = a_1 \frac{l_{11}}{A} + a_2 \frac{l_{21}}{A}$ となる.
- これで, d をよりシンプルな形で表すことができた.

0CTF 2017 Finals: Authentication&Secrecy

- ここからは, この d の別表現を用いて, 今度は N_2 についての式を変形していく.
  - $ed = 1 + k_2(N_2 - (p_2 + q_2 - 1))$ を次のように変形していく.
  - ただし, $l'_{11} = \frac{l_{11}}{A}$, $l'_{21} = \frac{l_{21}}{A}$.

$$\begin{aligned} ed &= 1 + k_2(N_2 - (p_2 + q_2 - 1)) \\ e(a_1 l'_{11} + a_2 l'_{21}) &= 1 + k_2(N_2 - (p_2 + q_2 - 1)) \\ ea_1 l'_{11} &= 1 + k_2(N_2 - (p_2 + q_2 - 1)) - ea_2 l'_{21} \end{aligned}$$

$x = k_2$, $y = -(p_2 + q_2 - 1)$, $z = a_2$ とおき, el'_{11} で剰余を取ると,

$$\begin{aligned} x(N_2 + y) - el'_{21} z + 1 &= ea_1 l'_{11} \\ &\equiv 0 \pmod{el'_{11}} \end{aligned}$$

0CTF 2017 Finals: Authentication&Secrecy

- うまい具合に 3 変数方程式となった.
  - 実は, 解の大きさも小さいことが示せる.
  - このことから, この方程式は 3 変数の Coppersmith's Method を用いて解くことができる.
- しかし, このままでは非線形項が残ってしまい効率がよくない.
  - そのため, $u = xy + 1$ とおいた Unravelled Linearization を用いてこれを一時的に線形化する.
  - その結果得られた多項式 f は以下の通りである.

$$f(u, x, z) = u + xN_2 - el'_{21}z$$

0CTF 2017 Finals: Authentication&Secrecy

- 後は, このスライドで述べていたように方程式を増やし, LLL によって小さいノルムを持つように変形する.
- 生成する多項式は多変数ということもあり解説したものとは異なる生成方法をとっている.
- m, t という 2 パラメータを与えているとき, i, j, k, l という変数について 2 種類の多項式

$$g_{i,j,k}(u, x, y, z) = x^i z^j f^k(u, x, z)(el'_{11})^{m-k}$$
$$h_{j,k,l}(u, x, y, z) = y^j z^{k-l} f^l(u, x, z)(el'_{11})^{m-l}$$

を生成する. ただし, $g_{i,j,k}(u, x, y, z)$ については $0 \leq i \leq m - k$, $0 \leq j \leq m - k - i$, $0 \leq k \leq m$, $h_{j,k,l}(u, x, y, z)$ については $1 \leq j \leq t$, $\left\lfloor \frac{m}{t} \right\rfloor \leq k \leq m$, $0 \leq l \leq k$.

0CTF 2017 Finals: Authentication&Secrecy

- このようにして 3 本の方程式を得, 消去式を用いてこれらを 1 変数方程式とすれば良いと考えられる.
  - しかし, 消去式の計算は時間がかかる傾向にあり, 3 変数ということも考えるとより高速に同じ解を持つ線形方程式の集合からシンプルな 1 変数方程式のみを取り出せるような手法が求められる.
- そこで Gröbner 基底 を用いる.

0CTF 2017 Finals: Authentication&Secrecy

- Gröbner 基底の理論は極めて応用範囲の広い理論であり, 時間の都合上概説のみとする.
  - 暗号理論的な側面について触れた日本語文献としては [辻井 08] 第 6 章 「多変数公開鍵暗号とグレブナ基底」をおすすめする.
- 簡潔に述べるのであれば, 連立方程式の解を求める手法の一つである.
  - もう少し踏み込んだ言い方をすれば, 与えられた連立方程式から, 解集合を変えないかつ必要最低限にシンプルな連立方程式を計算するものである.

0CTF 2017 Finals: Authentication&Secrecy

- ゆえに, LLL により得られた基底から Howgrave-Graham's Lemma の条件を満たしている多項式全ての集合を作り, その Gröbner 基底を計算すれば, 求めたい解を比較的少ない時間で得ることができると結論付けられる.
- 実際, この論文では消去式を用いずに Gröbner 基底を用いて計算し, 与える方程式の数も調整することで著者らの環境では 16 秒程度の時間で計算が終了することが示されている.

0CTF 2017 Finals: Authentication&Secrecy

- 今まで見てきた流れをアルゴリズムとして見なおす.

  Step 1. LLL アルゴリズムを用いて, 基底 $\begin{pmatrix} A & 0 \\ e & N_1 \end{pmatrix}$ により張られる格子のより小さい基底を取り出す.

  Step 2. Step 1. で得たベクトルから, 未知変数 a_1, a_2 に対する関係式 $d = a_1 l'_{11} + a_2 l'_{21}$ を満たす成分 l'_{11}, l'_{21} を取り出す.

  Step 3. Step 2. で得た l'_{11}, l'_{21} を用いて $\bmod (el'_{11})^m$ で $f(u, x, z)$ と同じ根を持つ多項式 $g_{i,j,k}(u, x, y, z)$, $h_{j,k,l}(u, x, y, z)$ を計算する.

  Step 4. $g_{i,j,k}(u, x, y, z)$, $h_{j,k,l}(u, x, y, z)$ の係数ベクトルから行列 B を構成する.

  Step 5. LLL アルゴリズムを用いて格子基底 B から Thm. 12 を満たす格子基底 B' を計算する.

0CTF 2017 Finals: Authentication&Secrecy

Step 6. B' の各ベクトルのうち, Howgrave-Graham's Lemma (Lem. 7) の条件を満たすベクトルを取り出す.

Step 7. Step 6. で得たベクトルを多項式へと変換し, Unravelled Linearization によって線形化された項を $u = xy + 1$ によって戻す.

Step 8. Step 7. で得た多項式の Gröbner 基底を計算する.

Step 9. Step 8. で得た基底が x, y, z の解を表していれば, その解を返す. 解がなければ, 解なしを出力する.

0CTF 2017 Finals: Authentication&Secrecy

- このアルゴリズムを実装することで, N_1, N_2, e から d が得られ, RSA を破ることができる.
  - CTF 中の問題リリースから論文検索, 実装, デバッグ時間も含めて 8〜10 時間程度.
- 今回解くのに利用した論文は購入必須だが, CTF 中は躊躇なく購入ボタンを押す.
  - 結果的に解法に結びつかなかったとしても, 後から読める論文が増えることになるので損はない.
  - 既に大学・研究機関・会社で論文誌の契約等がある場合には大いに活用すべき.

0CTF 2017 Finals: Authentication&Secrecy

- 蛇足
  - CTF Crypto の場合, 競技の一環としてコードを書くこと, 論文を多数短い時間で正確に意図をつかむために読むことが重要になる.
    - ⋆ そのため, 普通の文章の読み書き能力は高い方がよい.
  - 活字を読み慣れておくと論文を読む速度も上がりやすいため, 趣味として小説を読むのは実は競技的にも有効.

Workshop

Workshop

- 今回紹介した手法が掲載されている論文は 2018/02/22 時点で Springer によって有料販売されているものしか存在しない.
  - 手法については今まで解説したとおりの手法であるため, これを参考に実装して欲しい.

Workshop

- 実際に Coppersmith's Method の実装を書く際には, 得られた多項式が Howgrave-Graham's Lemma を満たしているかどうかをチェックすることはあまりない
  - 得られた多項式のうち最初 10 個を雑に取ってくるような実装を書くことが多い.
- また, 行の入れ替えによって行列式・格子は変わらないため綺麗に三角行列を作る必要はない.
  - 列についても同様. 単項式の並べ方は多少雑でもなんとかなる.

Workshop

- 理論的考察に自信がなければ, いくつかのパラメータを結果の目視確認を用いて Heuristic に設定してしまうのも手.
  - これはハマると大変時間を吸い取られるため, 実装のデバッグが完璧なときのみ用いる.
- 添字の範囲が $<$ で指定されているのか $\leq$ で指定されているのかのチェックは欠かさず行う.
  - 添字の数えはじめであったり, 横ベクトル/縦ベクトルの差等細かいところでミスが起きやすいため, 「これを実装する」と決めた論文は焦って実装せずに一度理論部分を読み通し, 紙の上で自分なりに整理することを心がける.

Workshop

- Python の `range/xrange` 関数の引数はミスしやすいので注意が必要.
  - $n \leq i \leq m$ ならば `range(n, m+1)`
  - 逆順 $(i = m, m-1, ..., n)$ の場合は `range(m, n-1, -1)`

Workshop

- SageMath において Gröbner 基底を用いる際の Tips
- 整数係数多項式の場合, Gröbner 基底を求めるのにメモリ・CPU を異様に消費するようになることがある
  - おそらく整数が環であって除算が使えないためと思われる.
- そのため. 除算が使えるように一旦対象となる多項式を有理数係数多項式に変換し, そのうえで Gröbner 基底を計算するようにするとよい.
  - 高速化の Tips としては, デフォルトのアルゴリズムである Singular を用いるものから giac 等他のアルゴリズムを用いるようにするというものもある.

Workshop

- 実際, この問題を解いている当時整数係数多項式 20 個による Gröbner 基底を計算しようとしたところ, AWS のメモリ 128GB インスタンスを 20 分ほどで使い切り, 強制終了された.
  - 有理数係数多項式へと変換することで, これは 1GB 以下に収まるようになった.
  - 実行時間も 2 分足らず程度まで改善されており, giac を用いることで更に高速になった.
- SageMath では, 以下 2 行を挟むだけでよい.

```
vs = map(str, f.parent().gens())
f = PolynomialRing(QQ, vs)(f).parent()
```

Workshop

- 消去式を用いる解法の場合，「格子基底の先頭からいくつの多項式について」と限定した考察が多いかと思う．
  - 実際には全ての整数方程式同士の組み合わせを試して成功すればよし，という実装をすると解ける確率が気持ち程度上がる．
- 最初から綺麗に書こうとは思わずに，ある程度忠実に実装してバグを減らすことは常々心がけていくべき．
  - テストパラメータが論文に示されている場合は，それを用いて assertion を書いて確かめていく余裕を持つ．

Workshop

- 私が当時書いたソルバは以下の URL に存在する.
  `https://github.com/elliptic-shiho/ctf-writeups/blob/gh-pages/content/ctf/2017/0CTF%20Finals/cr1000-AuthenticationSecrecy/solve.sage`
  - 少々汚い上に英語の誤りも存在するが, 当時のままということでご容赦願いたい.
  - Gröbner 基底の計算に giac を用いているが, 手元にはおそらく入っていないと思われるため, その場合'giac' の部分を削除して動かしていただきたい.

Workshop

- 今回解説したい内容については以上である.
  - もしも不明な点・誤り等あれば遠慮なく質問をお願いしたい.

参考文献 I

[Ale07] Alexander May.
Using LLL-Reduction for Solving RSA and Factorization Problems: A Survey.
LLL+25 Conference in honour of the 25th birthday of the LLL algorithm, 2007.

[BD99] Dan Boneh and Glenn Durfee.
Cryptanalysis of RSA with Private Key d Less than $N^{0.292}$, pp. 1–11.
Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

[Bon99] Dan Boneh.
Twenty years of attacks on the rsa cryptosystem.
NOTICES OF THE AMS, Vol. 46, pp. 203–213, 1999.

参考文献 II

[CFPR96] Don Coppersmith, Matthew K. Franklin, Jacques Patarin, and Michael K. Reiter.
Low-exponent RSA with related messages.
In *Advances in Cryptology - EUROCRYPT '96, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16, 1996, Proceeding*, pp. 1–9, 1996.

[Cop96a] Don Coppersmith.
Finding a small root of a bivariate integer equation; factoring with high bits known.
In *Advances in Cryptology - EUROCRYPT '96, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16, 1996, Proceeding*, pp. 178–189, 1996.

参考文献 III

[Cop96b] Don Coppersmith.
Finding a small root of a univariate modular equation.
In *Advances in Cryptology - EUROCRYPT '96, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16, 1996, Proceeding*, pp. 155–165, 1996.

[Her11] Mathias Herrmann.
Lattice-based Cryptanalysis using Unravelled Linearization.
PhD thesis, 2011.

[HG97] Nicholas Howgrave-Graham.
Finding small roots of univariate modular equations revisited, pp. 131–142.
Springer Berlin Heidelberg, Berlin, Heidelberg, 1997.

参考文献 IV

[HG01] Nick Howgrave-Graham.
Approximate integer common divisors.
In Joseph H. Silverman, editor, *Cryptography and Lattices*, pp. 51–66, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[Hin07] Hinek, M. Jason.
On the Security of Some Variants of RSA.
PhD thesis, 2007.

[HM09] Mathias Herrmann and Alexander May.
Attacking power generators using unravelled linearization: When do we output too much?
In Mitsuru Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, pp. 487–504, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

参考文献 V

[HM10] Mathias Herrmann and Alexander May.
Maximizing small root bounds by linearization and applications to small secret exponent rsa.
In Phong Q. Nguyen and David Pointcheval, editors, *Public Key Cryptography – PKC 2010*, pp. 53–69, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[JM06] Ellen Jochemsz and Alexander May.
A strategy for finding roots of multivariate polynomials with new applications in attacking rsa variants.
In Xuejia Lai and Kefei Chen, editors, *Advances in Cryptology – ASIACRYPT 2006*, pp. 267–282, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

参考文献 VI

[KO08] Noboru Kunihiro and Kazuo Ohta.
RSA 暗号に対する格子理論に基づく攻撃.
Bulletin of the Japan Society for Industrial and Applied Mathematics, 2008.

[LLL82] A. K. Lenstra, H. W. Lenstra, and L. Lovász.
Factoring polynomials with rational coefficients.
Mathematische Annalen, Vol. 261, No. 4, pp. 515–534, Dec 1982.

[May03] Alexander May.
New RSA Vulnerabilities Using Lattice Reduction Methods.
2003.

参考文献 VII

[NSS+17] Matus Nemec, Marek Sys, Petr Svenda, Dusan Klinec, and Vashek Matyas.
The Return of Coppersmith's Attack: Practical Factorization of Widely Used RSA Moduli.
In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pp. 1631–1648, New York, NY, USA, 2017. ACM.

[Pei13] Chris Peikert.
Lattices in Cryptography - Lecture 1: Mathematical Background, 2013.
http://web.eecs.umich.edu/~cpeikert/lic15/lec01.pdf.

参考文献 VIII

[PHL+17] Liqiang Peng, Lei Hu, Yao Lu, Jun Xu, and Zhangjie Huang.
Cryptanalysis of dual rsa.
Designs, Codes and Cryptography, Vol. 83, No. 1, pp. 1–21, Apr 2017.

[RSA78] Ronald L Rivest, Adi Shamir, and Leonard Adleman.
A method for obtaining digital signatures and public-key cryptosystems.
Communications of the ACM, Vol. 21, No. 2, pp. 120–126, 1978.

[辻井 08] 辻井 重男, 笠原 正雄, 有田 正剛, 境 隆一, 只木 孝太郎, 趙 晋輝, 松尾 和人.
暗号理論と楕円曲線 - 数学的土壌の上に花開く暗号技術.
森北出版, 2008.