セキュリティ・キャンプ2020 Ｌトラック 成果報告

ＬＩ 暗号解読チャレンジゼミ

回答ＩＤ 50

キャンプ期間中にやったこと一覧

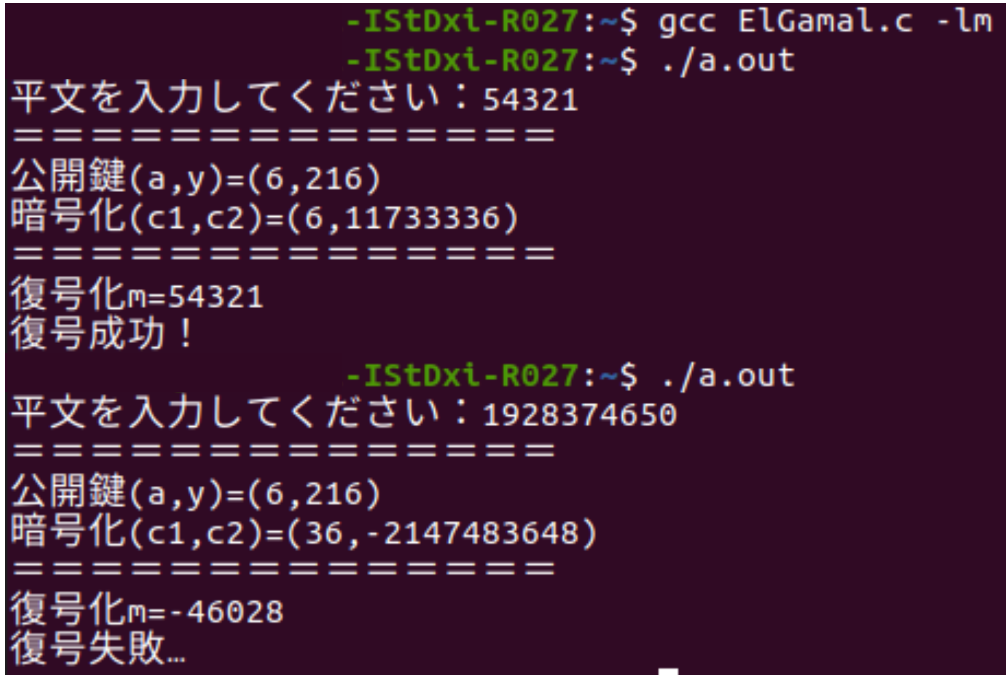
- ElGamal暗号
  1. 応募課題で実装したElGamal暗号の改良
  2. ElGamal暗号に対する選択暗号文攻撃の実装
  3. 離散対数高速計算アルゴリズム(Pohlig-Hellman)の実装

- Schmidt-Samoa暗号
  4. 初見の暗号(Schmidt-Samoa暗号)の提案論文を読む
  5. Schmidt-Samoa暗号の実装
  6. Schmidt-Samoa暗号の解読手法の実装

0 応募課題で実装したElGamal暗号

- ElGamal暗号の流れを実装 (使用言語 C)
  - $y(=a^x)$ は公開鍵，x は秘密鍵
  - 暗号化 $(c_1, c_2) = (a^r, my^r)$
  - 復号化 $m = c_2 / c_1^x$

▼上は正しく暗号化できた例
下は大きな値を入力してオーバーフローした例

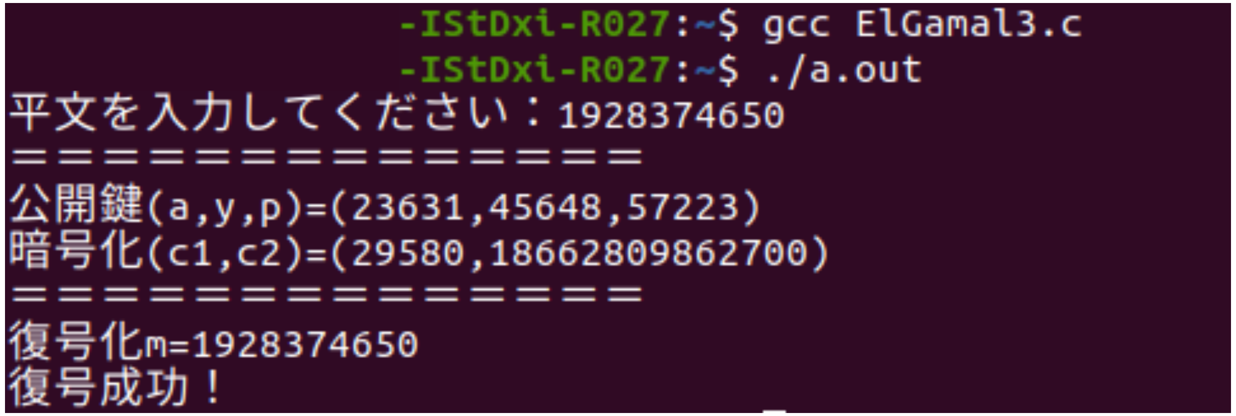


```
-IStDxi-R027:~$ gcc ElGamal.c -lm
-IStDxi-R027:~$ ./a.out
平文を入力してください：54321
==================
公開鍵(a,y)=(6,216)
暗号化(c1,c2)=(6,11733336)
==================
復号化m=54321
復号成功！
-IStDxi-R027:~$ ./a.out
平文を入力してください：1928374650
==================
公開鍵(a,y)=(6,216)
暗号化(c1,c2)=(36,-2147483648)
==================
復号化m=-46028
復号失敗…
```

- 値が大きくなるとオーバーフローする
  - 長い平文を入力できない
  - a や乱数 r の値を大きくできない

1 応募課題で実装したElGamal暗号の改良

- Powの計算にバイナリ法を使うことで高速化
  - 再帰関数を用いた実装
  - ビット操作を用いた実装
- moduloを用いて計算
- long型を使用



➢ 応募課題時の実装に比べて，かなり大きな値(intの上限程度)まで正しく暗号化できるようになった.

2 ElGamal暗号に対する選択暗号文攻撃の実装

- 選択暗号文攻撃(CCA)とは

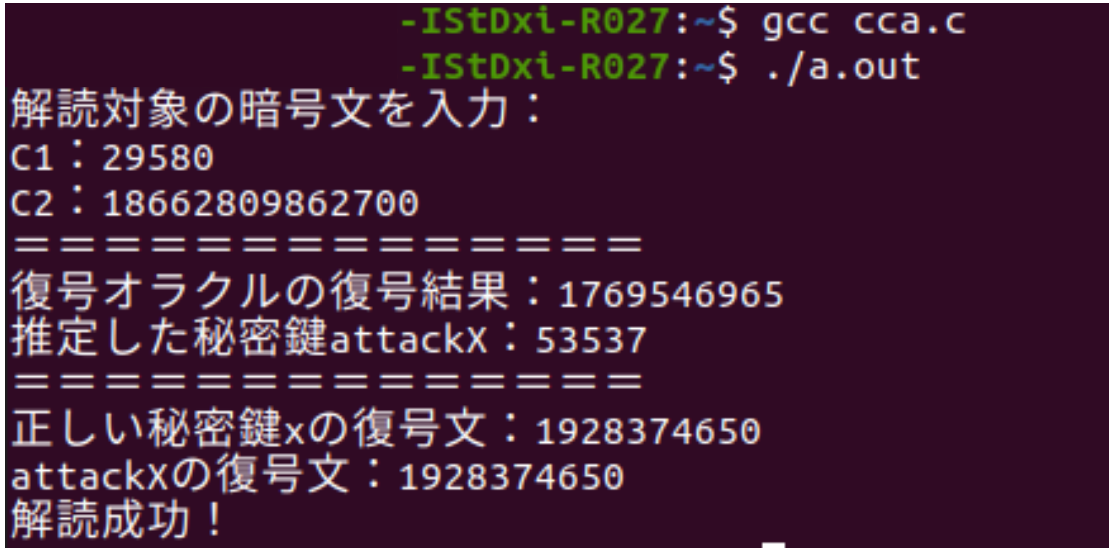
  攻撃者が任意に選択したある暗号文を正規ユーザに復号させることができた場合に，元の暗号文と得られた平文から暗号鍵を推測する攻撃.

- ElGamal暗号はIND-CPA安全だが，IND-CCA安全ではない.

  (応募課題で証明済)

2 ElGamal暗号に対する選択暗号文攻撃の実装

- アルゴリズムの流れ
  1. 攻撃者が適当に選んだ暗号文(c_1', c_2')について，正しい秘密鍵 x を用いて復号し，平文m′を得る.
  2. 攻撃者が秘密鍵 x′ を作り，$m' = \mathrm{Dec}(c_1', c_2', x')$となるようなx′を総当たりで求める.
  3. 2.で求めた鍵 x′ を用いて，解読対象の暗号文(c_1, c_2)から平文 m を入手する.



➢ スライド４ページの暗号文 (29580, 18662809862700) に対して，正しい平文を求めることができた.

3　離散対数高速計算アルゴリズムの実装

- ElGamal暗号の安全性は，離散対数問題の難しさに基づいている．
  - $y = a^x \pmod{p}$ について，p が素数で a が Z_p の生成元であるとき，y から x を求めることは困難．

- 離散対数を高速に解くアルゴリズムがいくつか存在する．
  - ベビーステップ・ジャイアントステップアルゴリズム
  - Pollardの ρ アルゴリズム
  - Pohlig-Hellmanアルゴリズム

など

- 今回は，Pohlig-Hellmanアルゴリズムを実装した．

3　離散対数高速計算アルゴリズムの実装

- Pohlig-Hellmanアルゴリズムの流れ

❶ $p-1 = q_0^{e_0}\ q_1^{e_1} \cdots q_k^{e_k}$ と素因数分解する．

❷ 各々の i (0〜k)について$x_i = x \bmod q_i^{e_i}$ を求める．

1. $n = (p-1) / q_i$とおく．

2. $y^n \bmod p$ を求める．

3. $y^n \bmod p = g^{na} \bmod p$ を満たすような a ($0 \leqq a < q_i$) を求める．

4. $\{y(g^{-1})^a\}^{(n/q)} \bmod p = g^{nb} \bmod p$ を満たすような b ($0 \leqq b < q_i$) を求める．
   - (逆元を計算するには，拡張ユークリッドの互除法やフェルマーの小定理を用いるとよい)

3　離散対数高速計算アルゴリズムの実装

5. $\{y(g^{-1})^{a+bq_i}\}^{(n/q_i{}^2)} \bmod p = g^{nc} \bmod p$ を満たすような c ($0 \leqq c < q_i$) を求める.

6. 以下同様に e の値まで繰り返す.

7. $x_i = a + bq_i + cq_i{}^2 + \cdots$ を求める.

❸ 連立合同式

$x \equiv x_1 \pmod{q_1{}^{e_1}}$　$\cdots$ $x \equiv x_k \pmod{q_k{}^{e_k}}$ を

中国剰余アルゴリズムを用いて計算する.

3　離散対数高速計算アルゴリズムの実装

- スライド４ページのElGamalアルゴリズムについて，Pohlig-Hellmanを実行した例

  - 総当たりで鍵を求める方法(fullSearch.c)と比較すると，(鍵の値が小さいので大きな差はないが，)Pohlig-Hellmanアルゴリズムの方が高速であることが実感できた.

```
-IStDxi-R027:~$ gcc pohlig2.c -lm
-IStDxi-R027:~$ ./a.out
【公開情報】(a,y,p)=(23631,53532,57223)
＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝
57222 = 28611 * 2^1
【x mod 2^1となるx_0】 1
28611 = 3179 * 3^2
【x mod 3^2となるx_1】 2
3179 = 289 * 11^1
【x mod 11^1となるx_2】 9
289 = 1 * 17^2
【x mod 17^2となるx_3】 253
＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝
推定鍵attackX＝55163
y(=a^x mod p)=53532
y(=a^attackX mod p)=53532
解読成功
＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝
実行時間は1.948000ミリ秒でした
-IStDxi-R027:~$ gcc fullsearch.c
-IStDxi-R027:~$ ./a.out
【公開情報】(a,y,p)=(23631,53532,57223)
＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝
推定鍵attackX＝55163
y(=a^x mod p)=53532
y(=a^attackX mod p)=53532
解読成功
＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝
実行時間は19.694000ミリ秒でした
```

上はPohlig-Hellmanアルゴリズムを用いてxを求めた例
下は総当たりでxを求めた例▶

4 Schmidt-Samoa暗号の提案論文を読む

- ElGamal暗号に一段落つけて，初見の暗号に挑戦．

- 論文を読んでみての所感
  - 予備知識がなかったので，新鮮な気持ちで読めた．
  - 数学的背景の理解に苦労した．
  - 英語の意味を履き違えたこと多数…

A New Rabin-type Trapdoor Permutation Equivalent to Factoring and Its Applications

Katja Schmidt-Samoa

Technische Universität Darmstadt, Fachbereich Informatik,
Hochschulstr. 10, D-64289 Darmstadt, Germany
samoa@informatik.tu-darmstadt.de

Abstract. Public key cryptography has been invented to overcome some key management problems in open networks. Although nearly all aspects of public key cryptography rely on the existence of trapdoor one-way functions, only a very few candidates of this primitive have been observed yet. In this paper, we introduce a new trapdoor one-way permutation based on the hardness of factoring integers of p^2q-type. We also propose a variant of this function with a different domain that provides some advantages for practical applications. To confirm this statement, we develop a simple hybrid encryption scheme based on our proposed trapdoor permutation that is CCA-secure in the random oracle model.

Keywords: trapdoor one-way permutations, EPOC, hybrid encryption, Tag-KEM/DEM framework

4 Schmidt-Samoa暗号の提案論文を読む

● Schmidt-Samoa暗号とは

- 2006年にK.Schmidt-Samoaが提案.
- Trapdoor one-way permutationを応用. (n(＝p^2q)からdを求めるのは困難)
- （提案した）Tag-KEM/DEMを用いることで，CCA安全なハイブリット暗号を構築.
  - ➢ KEM：<u>公開鍵暗号</u>を使って秘密鍵をカプセル化
  - ➢ DEM：<u>共通鍵暗号</u>によってデータを暗号化・復号
  - ➢ Tag：乱数値を設定することで，途中で改ざんされていないかチェック
- **IND-CCA安全**である.

5 Schmidt-Samoa暗号の実装

- Schmidt-Samoa暗号のアルゴリズム

【鍵生成】

- 2つの大きな素数 p, q を選択し，積 $N = p^2q$ を計算する．
- $d = N^{-1} \bmod \mathrm{lcm}(p-1, q-1)$ を計算する．
  - ➢ $(pk, sk) = (N, d)$ を出力

【暗号】

- 平文 m に対して $c = m^N \bmod N$ を計算する．
  - ➢ c を出力

【復号】

- 暗号文 c に対して $m = c^d \bmod pq$ を計算する．
  - ➢ m を出力

5 Schmidt-Samoa暗号の実装

- ElGamal暗号の実装からの変更点.
  - GMP(多倍長精度の演算ライブラリ)を用いることで，大きな桁同士の演算に対応.
  - 実行毎に鍵(素数の組)を生成するように改良.

```
               -IStDxi-R027:~$ gcc samoa2.c -lgmpxx -lgmp
               -IStDxi-R027:~$ ./a.out
平文を入力してください：4578747347247277846585765 7
(p,q)：(41579849451443,805833479872073)
鍵(N,d)： (13931925136412238739575475239346094712321 77,1240196316830544731669513 7921)
暗号： 111928748245881165534658221122945485171 6092
復号結果 4578747347247277846585765 7
復号成功
               -IStDxi-R027:~$ ./a.out
平文を入力してください：1237475464358448753485784 57
(p,q)：(305047401356759,1252064165231311)
鍵(N,d)： (11650947500340201961772616937829513112776 0191,42722795625877073218005139411)
暗号： 1049551269999320020317686456611688139453 47153
復号結果 1237475464358448753485784 57
復号成功
```

6 Schmidt-Samoa暗号の解読手法の実装

- かなり強力な暗号なので，一筋縄では行かない・・・

- 暗号の構造的欠陥を突くことで，条件付きで解読.
  - Enc : $c = m^N \bmod \mathbf{N}$ 　($N = p^2q$)
  - Dec : $m = c^d \bmod \mathbf{pq}$
  - 暗号と平文の法が一致していない.
    - $x^{Nd} \bmod pq$ と x が異なるのにも関わらず，$x^{Nd} \bmod pq$ と $x \bmod pq$ が一致するような x が存在する.

6 Schmidt-Samoa暗号の解読手法の実装

$pq = s$とおき，$0 < x < s$, $0 \leqq k < p$ とする.

二項定理より$(x + ks)^N \equiv x^N + x^{N-1}ks + x^{N-2}ks^2 + \dots + ks^N$
$\equiv x^N + x^{N-1}ks \pmod N$.

$\mathrm{Dec}(x^N + x^{N-1}ks) \equiv (x^N)^d + (x^N)^{d-1}(x^{N-1}ks) + \dots + (x^{N-1}ks)^d$
$\equiv (x^N)^d \equiv x \pmod s$.

逆に, ある $0 \leqq x < N$ に対し，$\mathrm{Dec}(\mathrm{Enc}(x)) \equiv x \pmod s$ であるとする.

このとき $y := \mathrm{Dec}(\mathrm{Enc}(x)) \mod s$ とおくと,

$\mathrm{Dec}(\mathrm{Enc}(x)) = y + ks$ なる k がただひとつ存在する.

$x' := x \bmod s$ とおく.

すると $x = x' + ls$なる $0 \leqq x' < s$ と l がそれぞれただひとつ存在する.

6 Schmidt-Samoa暗号の解読手法の実装

$x = x' + ls = y + ks$と $0 \leqq x' < s$, $0 \leqq y < s$ から $x' = y$.
すると $x' - y = ls - ks = (l - k)s$ から $l - k = 0$.

したがって, Dec(Enc(x)) = x なる x の全体は,
ちょうど$\{x + ks \mid 0 \leqq k < p\}$ですべて表現できる.

仮に攻撃者がこのような x を探せたとする.
このとき, $y := \mathrm{Dec}(\mathrm{Enc}(x))$ に対し $x - y = ks$ なる整数 k が存在する.
$0 \leqq x < N$ の範囲であれば $0 \leqq k < p$ であるから, $\gcd(x - y, N) = s$ となる.

6 Schmidt-Samoa暗号の解読手法の実装

例) x＝2とおくと，$2^{Nd} = 2 \bmod s$ より$2^{Nd} = 2 + ks$ (ただしk > 0).

- $2^{Nd} - 2 = ks$ より$\gcd(2^{Nd} - 2, N) = s$

この性質を用いて，N，d，cが分かっている状態から，pqおよびmを推測する.

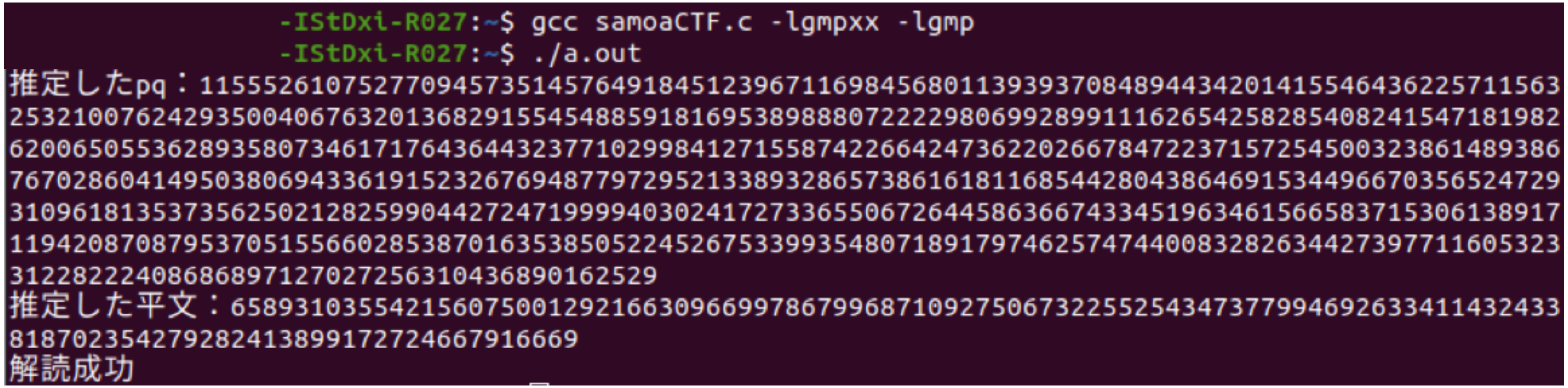
- 過去にCTFで出題された問題

N =
1043247381320041410146374847683905608886067988802815759655490071506964811427733496668055364700822779908166748193534341903
2853873326580815620415342289615384518300250417625376399851503449175840648276153565357638725399259899841170682075554987005
8537633073142835948979160031963607977702534421724847548583529868943792600395857814178437985795126497310921480391644570296
4047899447002191548965239884095802091798787988745415178578597206815122915438351636672681355517982410123569961242563194480
3993479617536781592102179935267383067965036605631630292874197322102475797932668910851788170475367584652399508612665925242
0639086982297246913756604517669027781335256339283884741767452925162741947981509786889099315321551548887709776027239948311
1013602618554846096893805108827211344467797072352072430591318485091677011261912992157762106672216780766138057271434398274
272110062768324672899209610001835423246735718435995420720133657793744810610657

d =
1057178776722242957842916753855467501296014721633795273753671282380034717930973417633919995039136595420875208083642871676
4534955589777442817628882903147770359477634126816043366897623719833012490437254689795395524177847532207416046976716256623
4358907740412805211247598698481640207956827199048419949848619731050860757808082106844953602176503247786535423423104239566
9107135286912986923321497836890327227720099744047167833122008855553438776110949981173079703992416402135538200434601404938
0340419077417191296265107510098383407125902103832288611466002878822909487254551605475770006826462802452439499579513546474
6570071201

6 Schmidt-Samoa暗号の解読手法の実装

c =
3433696867079169214970788514523959542520747536280248553265921485948282665678534977844489028399605998554572613876591749438
6486132479860740721019876253166550052366862767262338010028304770792796610171526619035547668020408128812410098300364955949
9979731342450364456322062273208742008750774795951290651360492364136674198126979777860541228166734597108964573061640243513
5714406892991664955413299908026024948966199955914853117790603536507001999810440583094060679352051967081780003706430186392
8200332830749288826325667210490527445364404202475143996724304914772476946478504332874164541921488573694048713906246378602
2104540986811119563549305718088453842093257734689508882969016290658885961965896726616361694426487326338043950416067501288
0659428625355600130372837615527974109807131865066718444180963953042340503425518499998670177709686335121785450448387667937
9573322245542186283569529914221900620132141788261777329136415389005047124410 4



```
-IStDxi-R027:~$ gcc samoaCTF.c -lgmpxx -lgmp
-IStDxi-R027:~$ ./a.out
推定したpq：1155526107527709457351457649184512396711698456801139393708489443420141554643622571156325321007624293500406763201368291554548859181695389888072222980699289911162654258285408241547181982620065055362893580734617176436444323771029984127155874226642473622026678472237157254500323861489386767028604149503806943361915232676948779729521338932865738616181168544280438646915344966703565247293109618135373562502128259904427247199994030241727336550672644586366743345196346156658371530613891711942087087953705155660285387016353850522452675339935480718917974625747440083282634427397711605323312282224086868971270272563104368901625 29
推定した平文：6589310355421560750012921663096699786799687109275067322552543473779946926334114324338187023542792824138991727246679166 69
解読成功
```

キャンプを振り返って

- 簡単な処理(forループや素因数分解など)だけでも多くの発見があった.
- 日常的にプログラムを書くのは疲れるけどいい体験だった.
- 少し講師に質問しすぎた… かも…
- コードが動くか否かが最優先で，理論的な部分やコードの効率性についてはあまり触れられなかった.

最後に

- キャンプ期間中に書いたコードはGitHubで公開しています.
  https://github.com/biyosh/SecCamp-L1

- ご清聴ありがとうございました.