知能情報演習：
公開鍵暗号による秘匿計算

九州工業大学　大学院情報工学研究院　知能情報工学研究系

坂本比呂志，井智弘

この演習の目的

1. 公開鍵暗号を用いた秘匿計算のしくみを理解
2. 秘匿計算ライブラリによる実習
3. さらなる発展課題への挑戦

予備知識

公開鍵暗号の基礎と
その応用としての完全
準同系暗号

公開鍵暗号

[**仮定**]

暗号化するメッセージMはあまり大きくない自然数とする。

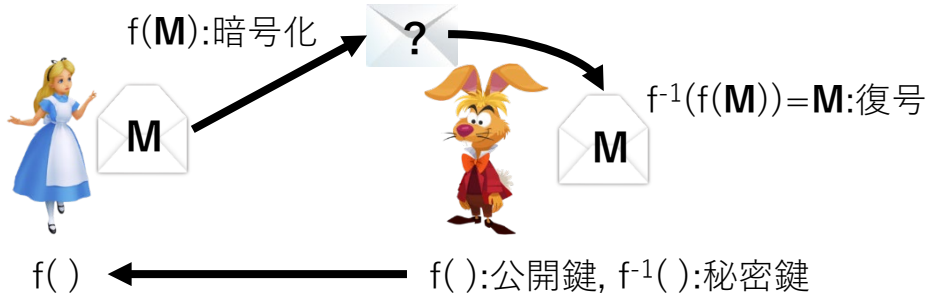
[**一般化**]

大きな整数や文字列はどう暗号化するか？

→ a = ‘10’, b= ‘11’, … のように文字と整数の対応をあらかじめ決めておけばよい。この対応は全員が共有しても安全性には無関係。

[**公開鍵暗号の概略**]

- アリスとボブが秘密裏にメッセージMをやり取りしたい
- ボブは暗号化のカギf()と復号のカギf^{-1}()を生成
  - f()を公開し、f^{-1}()を秘匿する
  - f^{-1}()からf()を推測することは困難であると仮定する
- アリスはボブの公開鍵f()でMを暗号化して送信
- ボブはf^{-1}()を使ってMを複合



一方向関数：
$f^{-1}()$の推測が困難（と信じられている）例

【$f^{-1}()$の推測が容易な問題】

GCD: Greatest Common Divisor 問題

与えられたいくつかの整数の最大公約数（GCD）を求める問題

ユークリッド互除法を用いて高速に計算可能

したがってGCDとその逆関数はいずれも簡単に解ける
（以下の等式の左右のどちらを入力としても解を求めることは容易，右から左を求めることがGCD）

$$c_1 = pq_1 = 101 * 10 = \underline{1010}$$
$$c_2 = pq_2 = 101 * 2 = \underline{202}$$
$$c_3 = pq_3 = 101 * 5 = \underline{505}$$
$$\vdots$$

【近似GCD問題】

GCDの問題に小さな（1ビットの）ノイズを混ぜてみる

$$c_1 = m_1 + pq_1 = 1 + 101 * 10 = \underline{1011}$$
$$c_2 = m_2 + pq_2 = 0 + 101 * 2 = \underline{202}$$
$$c_3 = m_3 + pq_3 = 1 + 101 * 5 = \underline{506}$$
$$\vdots$$

この問題は、右から左の計算（近似GCD）が計算困難であると信じられている

Dijk暗号：
近似GCDに基づく公開鍵暗号

【Dijk（ダイク）暗号】

（秘密鍵）p：十分大きな奇数

（公開鍵）：ランダムなq_i, r_i から $x_i = pq_i + r_i$ を計算（ i= 1,2,…,k）して、$x_1 > x_2 > \ldots > x_k$とソートする。このとき、x_1が奇数かつ$x_1 \bmod p = r_1$が偶数となるようにする。（これを満たすまでx_1～ x_kを取り直す）

このようにして生成した$x_1, x_2, \ldots, x_k$が公開鍵

【暗号化】

$$m \in \{0, 1\}: \text{message}$$

$$r \in \mathrm{N}: \text{noise} \quad (r < p)$$

$$\Rightarrow c = (m + 2r + 2 \sum_{1 \le i \le k} x_i) \mod x_1$$

Dijk暗号の復号　$c = (m + 2r + 2 \sum_{1 \le i \le k} x_i) \mod x_1$

暗号文cが与えられたとき、(c mod p) mod 2を計算する

$$c = (m + 2r + 2 \sum_{1 \le i \le k} x_i) \mod x_1 \Leftrightarrow (m + 2r + 2 \sum_{1 \le i \le k} x_i) = ex_1 + c$$

$$\Leftrightarrow c = m + 2r + 2 \sum_{1 \le i \le k} x_i - ex_1$$

$$\Rightarrow (c \mod p) = m + 2r + 2 \sum_{1 \le i \le k} r_i - er_1 \quad (x_i = pq_i + r_i)$$

$$\Rightarrow (c \mod p) \mod 2 = m \quad (r_1 \mod 2 = 0)$$

※注1：ここで、mは1ビット、r_1は偶数であることに注意するとc→mと復号できる
※注2：制約としてx_1は奇数である必要がある。これはなぜか考えてみよう。

Dijk暗号の準同型性

Dijk暗号は定義より以下の(1)と(2)の性質（加法および乗法の準同型性）を満たす。

ただし、右辺の+はビットのORで、*はビットのANDを意味する。

$$c_1 + c_2 = (m_1 + m_2 \mod p) \mod 2 \qquad (1)$$

$$c_1 * c_2 = (m_1 * m_2 \mod p) \mod 2 \qquad (2)$$

$$c = m + 2r + 2\sum_{1 \le i \le k} x_i - ex_1 \Rightarrow (c \mod p) = m + 2r + 2\sum_{1 \le i \le k} r_i - er_1 \quad (x_i = pq_i + r_i)$$

$$\Rightarrow (c \mod p) \mod 2 = m \quad (r_1 \mod 2 = 0)$$

これは、暗号文の和（積）が、平文の和（積）の暗号文と等しいことを意味する。

(1)と(2)を満たす暗号を完全準同系暗号という。

準同型暗号による秘匿計算

【加算器の秘匿計算】

整数x,yのi番目のビットをx_i,y_iとする。

s_iをx+yのi番目のビット、ciをx+yのi番目の繰り上がりビットとすると、x+y（整数の加算）を以下のビット演算で計算できる。ただし(c_0=0)。

$$s_i = x_i + y_i + c_i$$

$$c_{i+1} = (x_i + c_i) \cdot (y_i + c_i) + c_i$$

したがって、x,yの各ビットを暗号化した状態から、x+yの暗号文を計算できる、すなわち暗号化したまま加算できる。これを利用して、乗算器やその他の様々な計算を秘匿計算できる。

TFHEによる秘匿計算の実習

【TFHE】最速の完全準同系暗号

TFHE

- 完全準同型暗号の一つ
- 元論文
  Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, Malika Izabachène,
  Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds,
  ASIACRYPT (1) 2016: 3-33
- それまで激重だった bootstrapping が実用的な速度に
- 論文著者によって実装されたライブラリ（TFHE library）
  https://tfhe.github.io/tfhe/
- TFHE library を利用するのに便利な機能を追加したライブラリ
  （坂本研2021年卒研生作）
  https://github.com/hiroshi-kyutech/TFHE-tool

秘匿計算の流れ（単純な委託計算の場合）

- 公開鍵と秘密鍵のペアを作成
- 手持ちのデータを暗号化

暗号化したデータと公開鍵を送る

- 暗号化されたまま秘匿計算

暗号化された計算結果を返す

- 暗号化された計算結果を復号化

サンプルプログラムの構成

- 公開鍵と秘密鍵のペアを作成
- 手持ちのデータを暗号化

encryptor

暗号化したデータと公開鍵を送る

calculator

- 暗号化されたまま秘匿計算

暗号化された計算結果を返す

- 暗号化された計算結果を復号化

decryptor

サンプルプログラムの構成

- 公開鍵と秘密鍵のペアを作成
- 手持ちのデータを暗号化

encryptor

暗号化したデータと公開鍵を送る

送受信は省略
全てローカルで行う

calculator

- 暗号化されたまま秘匿計算

暗号化された計算結果を返す

- 暗号化された計算結果を復号化

decryptor

サンプルコードの解説1

単純な秘匿計算をやってみよう

simple_example/encryptor.cpp

```cpp
#include <tfhe_libex/tfhe_libex.hpp>

int main() {
    // seed は秘密鍵を生成するために使われる擬似乱数の種（整数配列）
    // これはサンプルコードなので適当な値で固定しているが，
    // 実際に秘匿データを扱うときはまともな種の与え方をする必要がある
    uint32_t seed[] = {314, 1592, 657};
    TFHEConfig::setSeed(seed, 3);
    // 推定強度128bit以上の暗号強度で秘密鍵を作る
    TFHEKeySet::create_new_keyset(128);

    TFHEKeySet::save_secret_key("secret.key");
    printf("secret key saved to \"secret.key\".\n");
    TFHEKeySet::save_cloud_key("cloud.key");
    printf("cloud key saved to \"cloud.key\".\n");

    int a = 28;
    int b = -10;
    TFHEInteger<8> integer1(a); // aを8bit整数として暗号化したオブジェクトを生成
    TFHEInteger<8> integer2(b); // bを8bit整数として暗号化したオブジェクトを生成
    TFHEInteger<16> integer3(a); // aを16bit整数として暗号化したオブジェクトを生成
    TFHEInteger<16> integer4(b); // bを16bit整数として暗号化したオブジェクトを生成

    integer1.save("cloud_integer.data", 1);
    printf("ciphertext %d (8bit integer) saved to \"cloud_integer.data\" with tag 1.\n", a);
    integer2.save("cloud_integer.data", 2);
    printf("ciphertext %d (8bit integer) saved to \"cloud_integer.data\" with tag 2.\n", b);
    integer3.save("cloud_integer.data", 3);
```

ライブラリをインクルード
エラーが出る場合はインクルードパスが通っているか確認

秘密鍵と共通鍵を作る
本演習ではおまじないと思っていてよい

実際に秘匿データを扱うときは，セキュリティ第一なので，理解して使いましょう

秘密鍵と共通鍵の保存

暗号化した整数オブジェクトを生成

暗号化した整数を保存

第二引数で整数タグをつける（読み込み時に使う）

simple_example/calculator.cpp

```cpp
#include <tfhe_libex/tfhe_libex.hpp>

// 暗号化されたデータをファイルから読み込み計算する
int main() {
    TFHEKeySet::load_cloud_key("cloud.key");
    printf("cloud key loaded from \"cloud.key\".\n");
    TFHEConfig::safe_mode = false;

    // cloud_integer.data に保存されている暗号化データを読み込む
    TFHEInteger<8> integer1("cloud_integer.data", 1);
    printf("ciphertext loaded from \"cloud_integer.data\" with tag 1.\n");
    TFHEInteger<8> integer2("cloud_integer.data", 2);
    printf("ciphertext loaded from \"cloud_integer.data\" with tag 2.\n");
    TFHEInteger<16> integer3("cloud_integer.data", 3);
    printf("ciphertext loaded from \"cloud_integer.data\" with tag 3.\n");
    TFHEInteger<16> integer4("cloud_integer.data", 4);
    printf("ciphertext loaded from \"cloud_integer.data\" with tag 4.\n");

    TFHEInteger<8> integer5 = integer1 + integer2;
    printf("generated ciphertext for integer1 + integer2.\n");
    TFHEInteger<16> integer6 = integer3 * integer4;
    printf("generated ciphertext for integer3 * integer4.\n");

    integer5.save("cloud_integer.data", 5);
    integer6.save("cloud_integer.data", 6);
    printf("ciphertexts saved to \"cloud_integer.data\" with tags 5 and 6.\n");

    return 0;
}
```

共通鍵を読み込む

safe_mode = true; にするとエラー検出機能がオンになる（その代わり遅くなる）

データを読み込む

2つ目の引数で保存時に使用した整数タグ指定する

秘匿計算
演算子がオーバーロードされているので普通の計算を行う感覚で使える

計算結果を保存

simple_example/decryptor.cpp

```cpp
#include <tfhe_libex/tfhe_libex.hpp>

// 計算結果を復号化して表示する
int main() {
    TFHEKeySet::load_secret_key("secret.key");
    printf("secret key loaded from \"secret.key\"\n");

    TFHEInteger<8> result1("cloud_integer.data", 5);
    TFHEInteger<16> result2("cloud_integer.data", 6);
    printf("ciphertext loaded from \"cloud_integer.data\" with tag 5.\n");
    printf("ciphertext loaded from \"cloud_integer.data\" with tag 6.\n");

    // 復号化して結果を表示する
    printf("decrypted result1: %d\n", result1.decrypt());
    printf("decrypted result2: %d\n", result2.decrypt());

    return 0;
}
```

秘密鍵を読み込む

データを読み込む

復号化して表示

演習1

サンプルコードを改変し，足し算と掛け算以外の演算を行ってください．

参考：

- オンラインドキュメント
  https://hiroshi-kyutech.github.io/TFHE-tool/public/
- テストコード
  https://github.com/hiroshi-kyutech/TFHE-tool/tree/main/src/test

サンプルコードの解説2

配列の使い方とソート

insert_sort/encryptor.cpp

```cpp
#include <tfhe_libex/tfhe_libex.hpp>

int main() {
    uint32_t seed[] = {314, 1592, 657};
    TFHEConfig::setSeed(seed, 3);
    TFHEKeySet::create_new_keyset(128);

    TFHEKeySet::save_secret_key("secret.key");
    TFHEKeySet::save_cloud_key("cloud.key");

    std::vector<int> vec1{5, 2, 3, 8, 21, 1, 13};
    // 中身が暗号化された8ビット整数配列を作る
    TFHEArray<TFHEInt8> array1(vec1);

    array1.save("cloud_array.data", 1);
    printf("ciphertext (array1 [5, 2, 3, 8, 21, 1, 13]) saved with tag 1.\n");

    return 0;
}
```

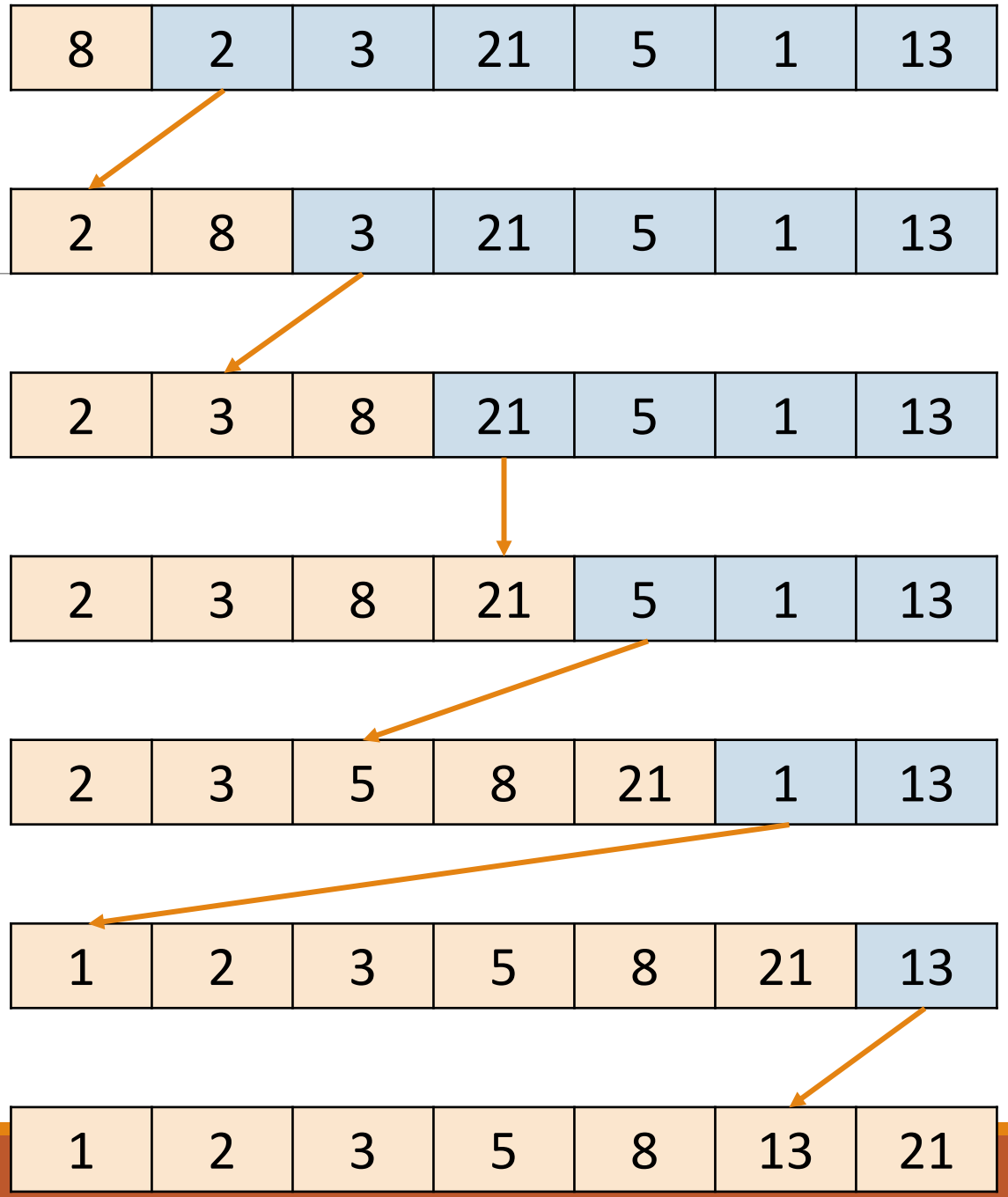
鍵生成のおまじない

std::vector<int> を引数に
暗号化された整数配列を生成

暗号化した配列を保存

挿入ソート

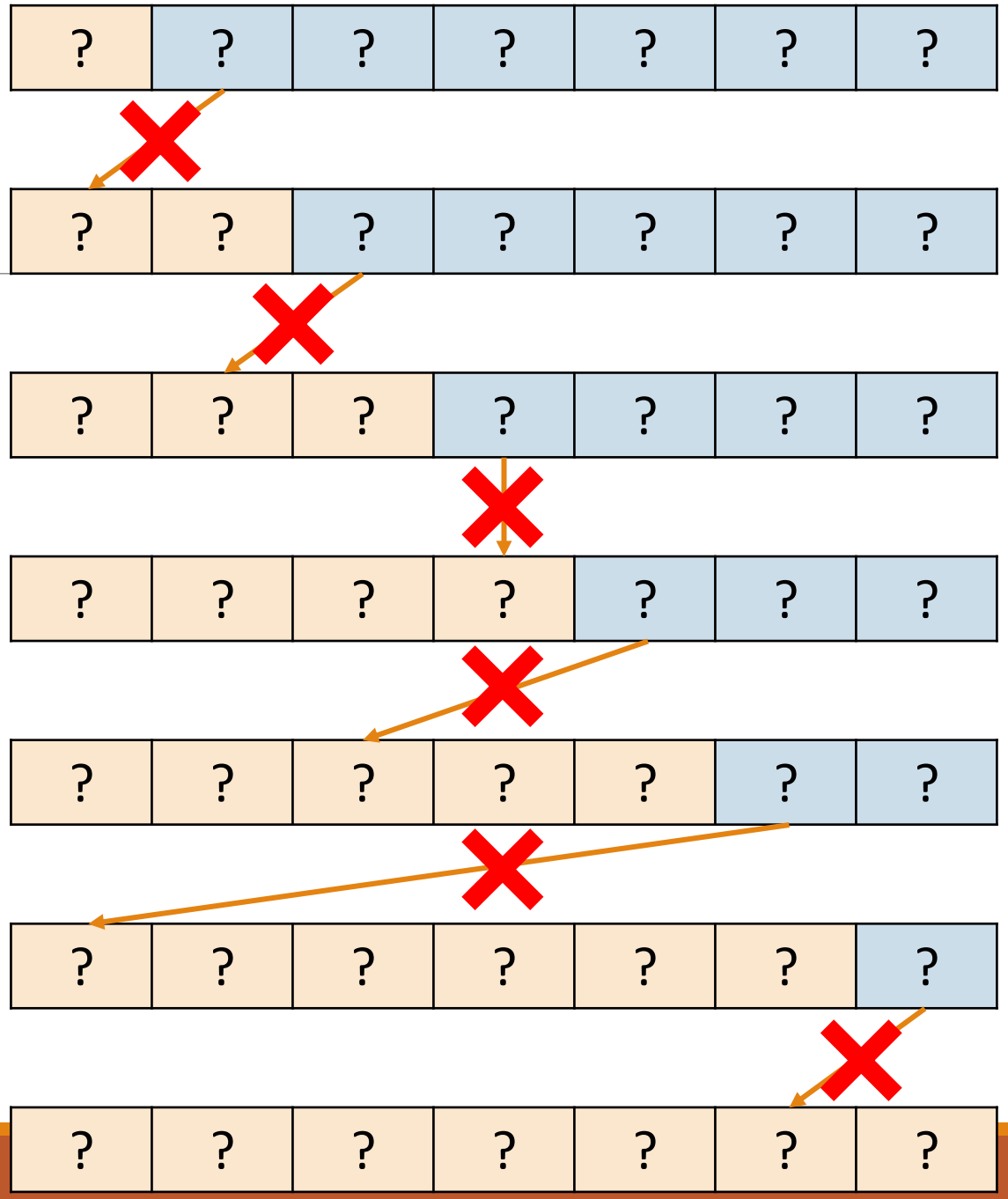
- 配列 x の位置を昇順に見る
- 位置 i を見るとき，
  - 前提：$x[0..i-1]$ はソートされた配列になっている
  - 仕事：$x[0..i]$ がソートされた配列になるように $x[i]$ を $x[0..i-1]$ 中の適切な位置に挿入する



秘匿計算での挿入ソートの問題点

- 配列 x の位置を昇順に見る
- 位置 i を見るとき，
  - 前提：$x[0..i-1]$ はソートされた配列になっている
  - 仕事：$x[0..i]$ がソートされた配列になるように $x[i]$ を $x[0..i-1]$ 中の適切な位置に挿入する

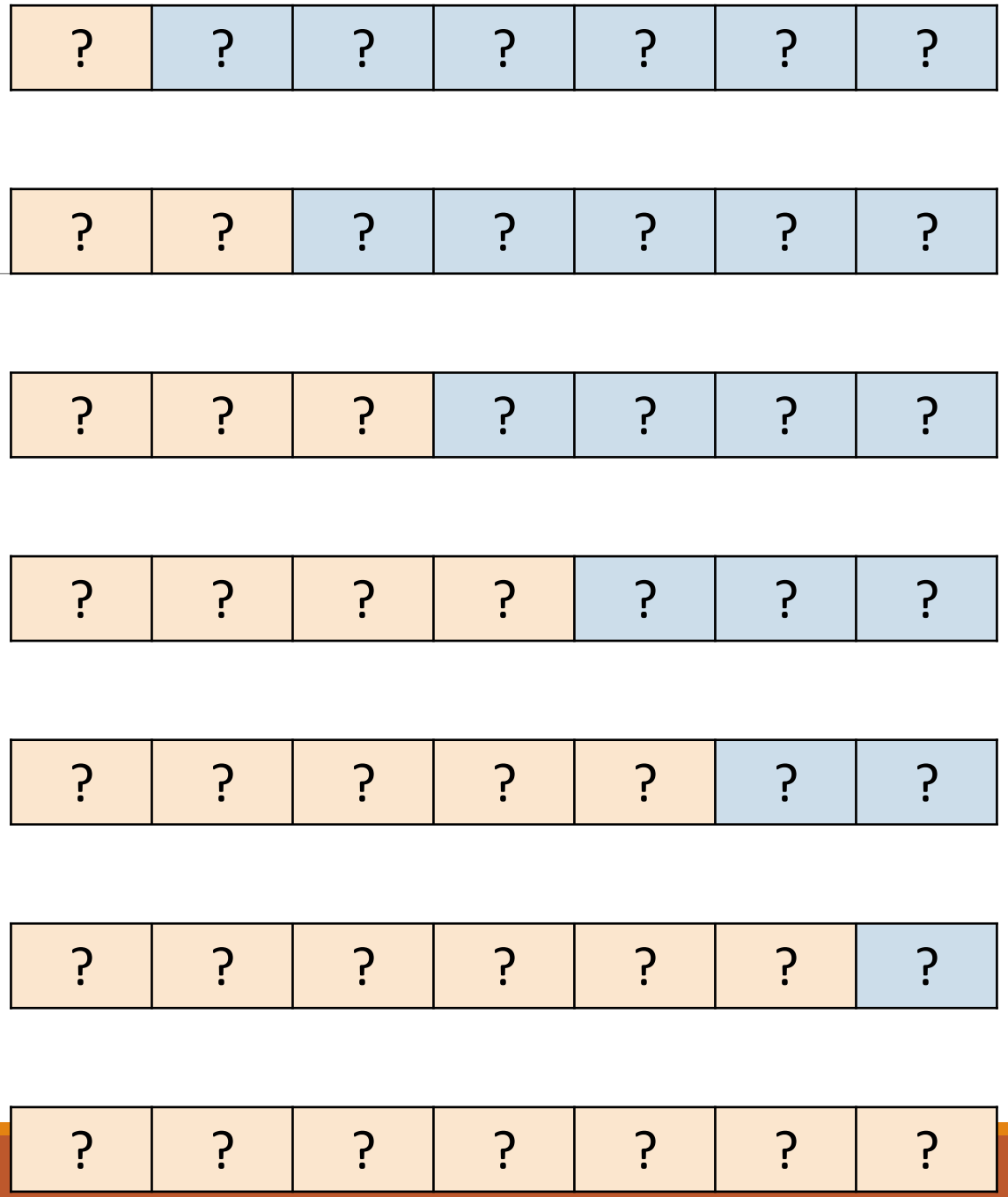
問題点
秘匿計算では，「適切な位置」を知ることはできない
それがわかると元データの並びに関する情報が漏れていることになるため



秘匿計算での挿入ソート

- 配列 x の位置を昇順に見る
- 位置 i を見るとき，
  - 前提：$x[0..i-1]$ はソートされた配列になっている
  - 仕事：$x[0..i]$ がソートされた配列になるように要素の交換を繰り返す（バブルソートみたいに）

ポイント
要素の大小に基づいて要素を秘密裏に交換することは可能



insert_sort/calculator.cpp の挿入ソート実装部

```
void insert_sort(TFHEArray<TFHEInt8> &x) {
  // .shape で多次元配列の各次元のサイズが取得できる
  // 今回 x は一次元配列なので，x.shape[0] で配列の長さを取得する
  const int n = x.shape[0];
  for (int i = 1; i < n; ++i) {
    for (int j = i; j > 0; --j) {
      // x[j-1] と x[j] を比較し x[j-1] < x[j] になるように秘密裏に交換する
      TFHEBool result = x.at(j-1) > x.at(j);
      TFHEInt8 tmp = x.at(j-1);
      // mux(<bool値の暗号>, <値Xの暗号>, <値Yの暗号>) は b が true なら
      // <値Xの暗号'> を返し，b が false なら <値Yの暗号'> を返す
      // <値Xの暗号> と <値Xの暗号'> はどちらも値Xの暗号だが，
      // 見た目は変化しているので，復号できない人からはその同一性が判断できない
      x.at(j-1) = mux(result, x.at(j), x.at(j-1)); // x.at(j) と x.at(j-1) の小さい方を x.at(j-1) に入れる
      x.at(j) = mux(result, tmp, x.at(j)); // x.at(j) と x.at(j-1) の大きい方を x.at(j) に入れる
    }
  }
}
```

このループで $x[1..i]$ がソートされた配列になるように交換を繰り返す

要素の大小情報を暗号化したまま取得

ライブラリのマルチプレクサ関数 mux を使って秘密裏に交換
関数 mux(b, X, Y) の中身は
b * X + (1 – b) * Y
という計算を暗号化したまま行っている

演習2

暗号化された配列上で何らかの計算を行うプログラムを作ってください

例:

- 合計値，中央値，最大値，最大値の位置などを求める
- 二つの配列の内積を求める