```
#include <bits/stdc++.h>
using namespace std;

const long long PRIME = 230584300921369351LL; prime)

// Modular exponentiation
long long modPow(long long a, long long b, long long p = PRIME) {
    long long res = 1;
    a %= p;
    while (b > 0) {
        if (b & 1) res = (__int128)res * a % p;
        a = (__int128)a * a % p;
        b >>= 1;
    }
    return res;
}

// Modular inverse
long long modInverse(long long a, long long p = PRIME) {
    return modPow(a, p - 2, p);
}

// Lagrange interpolation
long long lagrangeInterpolation(long long target, const vector<pair<long long,long long>>&
shares, long long p = PRIME) {
    long long total = 0;
```

```
    int k = shares.size();
    for (int i = 0; i < k; i++) {
        long long xi = shares[i].first;
        long long yi = shares[i].second;
        long long num = 1, den = 1;
        for (int j = 0; j < k; j++) {
            if (i == j) continue;
            long long xj = shares[j].first;
            num = (__int128)num * (target - xj + p) % p;
            den = (__int128)den * (xi - xj + p) % p;
        }
        long long li = (__int128)num * modInverse(den, p) % p;
        total = (total + yi * li) % p;
    }
    return (total + p) % p;
}

// Convert value string in given base to decimal
long long baseToDecimal(const string &s, int base) {
    long long result = 0;
    for (char c : s) {
        int digit;
        if (isdigit(c)) digit = c - '0';
        else if (isalpha(c)) digit = 10 + (tolower(c) - 'a');
        else continue;
        if (digit >= base) throw invalid_argument("Digit out of range for base");
```

```
        result = result * base + digit;
    }
    return result;
}

int main() {

    int n = 10, k = 7;

    vector<pair<long long,long long>> shares = {
        {1, baseToDecimal("13444211440455345511", 6)},
        {2, baseToDecimal("aed7015a346d635", 15)},
        {3, baseToDecimal("6aeeb69631c227c", 15)},
        {4, baseToDecimal("e1b5e05623d881f", 16)},
        {5, baseToDecimal("31603451457365262O673", 8)},
        {6, baseToDecimal("21222122011220022211202002100110202202O0", 3)},
        {7, baseToDecimal("20120221122211000100210021102001201112121", 3)},
        {8, baseToDecimal("2022055433533024O002224253", 6)},
        {9, baseToDecimal("45153788322a1255483", 12)},
        {10, baseToDecimal("1101613130313526312514143", 7)}
    };

    // Reconstruct majority secret
    map<long long, int> freq;
    vector<int> idx(n);
    fill(idx.begin(), idx.begin() + k, 1);
```

```
do {
   vector<pair<long long,long long>> subset;
   for (int i = 0; i < n; i++) if (idx[i]) subset.push_back(shares[i]);
   long long secret = lagrangeInterpolation(0, subset);
   freq[secret]++;
} while (prev_permutation(idx.begin(), idx.end()));

// Find most frequent secret
long long true_secret = -1;
int best = -1;
for (auto &kv : freq) {
   if (kv.second > best) {
      best = kv.second;
      true_secret = kv.first;
   }
}

cout << "Recovered Secret: " << true_secret << "\n";

// Detect wrong shares
vector<pair<long long,long long>> wrong;
for (auto &sh : shares) {
   vector<pair<long long,long long>> others;
   for (auto &s : shares) if (s != sh) others.push_back(s);
   if ((int)others.size() >= k-1) {
```

```
            vector<pair<long long,long long>> testSubset(others.begin(), others.begin() + (k-1));
            testSubset.push_back(sh);
            long long expected = lagrangeInterpolation(sh.first, testSubset);
            if (expected != sh.second) wrong.push_back(sh);
        }
    }

    cout << "Wrong Shares: ";
    if (wrong.empty()) cout << "None\n";
    else {
        for (auto &sh : wrong) cout << "(" << sh.first << "," << sh.second << ") ";
        cout << "\n";
    }

    return 0;
}
```