First assignment (2021/2022, this is the one you need to do, due 09-01-2022)
Merkle-Hellman cryptosystem (part 1)

In 1978, Merkle and Hellman proposed an interesting way to cipher information, based on an NP-complete problem (the knapsack problem). It turned out that this cipher is insecure. The techniques used to break it are outside the scope of this course. Below, we describe only one simplification of the cipher, where the knapsack problem is replaced by a subset-sum problem. (Hint: one of the techniques used to solve problem 2 of the homework of lesson P.02 may be useful here). In this assignment you will attempt to solve small instances of this problem.

The main idea of the Merkle-Hellman cryptosystem is to construct a subset-sum problem that is trivial to solve using a greedy algorithm, kept secret, and then to disguise it as a hard to solve problem, that is published. It is thus a public-key cryptosystem. The following information is kept secret:

- A super-increasing sequence $\boldsymbol{W} = (\boldsymbol{w_1}, \boldsymbol{w_2}, \ldots, \boldsymbol{w_n})$ of $\boldsymbol{n}$ positive integers. In a super-increasing sequence we have $\boldsymbol{w_k} > \sum_{i=1}^{k-1} \boldsymbol{w_i}$ for $\boldsymbol{2 \leqslant k \leqslant n}$.
- A modulus $\boldsymbol{m}$ such that $\boldsymbol{m > \sum_{i=1}^{n} w_i}$.
- A scrambling integer $\boldsymbol{a}$ such that $\boldsymbol{\gcd(a, m) = 1}$.

Let $\boldsymbol{W' = (w'_1, w'_2, \ldots, w'_n)}$, where $\boldsymbol{w'_i = (a w_i) \bmod m}$, for $\boldsymbol{1 \leqslant i \leqslant n}$. Recall that $\boldsymbol{a \bmod b}$ denotes the remainder of the division of $\boldsymbol{a}$ by $\boldsymbol{b}$. The following information is published:

- The sequence $\boldsymbol{P = (p_1, \ldots, p_n)}$, which is the sequence $\boldsymbol{W'}$ sorted in increasing order.

To cipher the $\boldsymbol{n}$ bits $\boldsymbol{B = (b_1, \ldots, b_n)}$ compute and send $\boldsymbol{C = \sum_{k=1}^{n} b_k p_k}$. Knowing $\boldsymbol{W}$, $\boldsymbol{m}$, and $\boldsymbol{a}$, it is easy to decode what was sent. Without that information, it is necessary to solve the hard subset-sum problem (see example in the next page).

Merkle-Hellman cryptosystem (part 2, example)

The following example show the Merkle-Hellman cryptosystem in action.

- Secret data: $\boldsymbol{W} = (\mathbf{1}, \mathbf{3}, \mathbf{5}, \mathbf{12}, \mathbf{22}, \mathbf{47})$, $\boldsymbol{m} = \mathbf{100}$, and $\boldsymbol{a} = \mathbf{13}$.
- Intermediate data: $\boldsymbol{W'} = (\mathbf{13}, \mathbf{39}, \mathbf{65}, \mathbf{56}, \mathbf{86}, \mathbf{11})$.
- Published data: $\boldsymbol{P} = (\mathbf{11}, \mathbf{13}, \mathbf{39}, \mathbf{56}, \mathbf{65}, \mathbf{86})$. Note that $\mathbf{11}$ is associated to $\boldsymbol{w}_6$, $\mathbf{13}$ to $\boldsymbol{w}_1$, $\mathbf{39}$ to $\boldsymbol{w}_2$, $\mathbf{56}$ to $\boldsymbol{w}_4$, $\mathbf{65}$ to $\boldsymbol{w}_3$, and $\mathbf{86}$ to $\boldsymbol{w}_5$.
- Unencrypted message: $\boldsymbol{B} = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_6) = (\mathbf{0}, \mathbf{0}, \mathbf{1}, \mathbf{1}, \mathbf{0}, \mathbf{1})$.
- Encrypted message: $\boldsymbol{C'} = \mathbf{0} \times \mathbf{11} + \mathbf{0} \times \mathbf{13} + \mathbf{1} \times \mathbf{39} + \mathbf{1} \times \mathbf{56} + \mathbf{0} \times \mathbf{65} + \mathbf{1} \times \mathbf{86} = \mathbf{181}$.

Knowing the secret information it is easy to decipher the message:

- Compute $\boldsymbol{C} = (\boldsymbol{C'a}^{-1}) \bmod \boldsymbol{m}$. Since $(\boldsymbol{w'_i a}^{-1}) \bmod \boldsymbol{m} = \boldsymbol{w}_i$, this transforms the hard problem into the easy problem. Because $\mathbf{13}^{-1} \bmod \mathbf{100} = \mathbf{77}$ — this is easy to check: $(\mathbf{13} \times \mathbf{77}) \bmod \mathbf{100} = \mathbf{1}$ — we get $\boldsymbol{C} = (\mathbf{181} \times \mathbf{77}) \bmod \mathbf{100} = \mathbf{37}$.
- Now the problem is easy: which elements of $\boldsymbol{W}$ sum to $\mathbf{37}$? $\mathbf{47}$ is clearly out, but $\mathbf{22}$ must be in. $\mathbf{37} - \mathbf{22} = \mathbf{15}$, so $\mathbf{12}$ must also be in. $\mathbf{15} - \mathbf{12} = \mathbf{3}$, so $\mathbf{3}$ is also in. So, $\mathbf{37} = \mathbf{22} + \mathbf{12} + \mathbf{3}$.
- To finish decoding, observe that $\mathbf{22} = \boldsymbol{w}_5$ is associated to $\mathbf{86} = \boldsymbol{p}_6$, $\mathbf{12} = \boldsymbol{w}_4$ is associated to $\mathbf{56} = \boldsymbol{p}_4$, and $\mathbf{3} = \boldsymbol{w}_2$ is associated to $\mathbf{39} = \boldsymbol{p}_3$, and so the non-zero bits are $\boldsymbol{b}_3$, $\boldsymbol{b}_4$, and $\boldsymbol{b}_6$.

Of course, in your written assignment you will not be able to do this last part, because you will not be given the secret information. You with have to solve the hard subset-sum problem.

Merkle-Hellman cryptosystem (part 3, what is given)

In the archive `A01.tgz` you will find the problem instances you need to solve. Each student will have her/his own problem instances, stored in a `.h` file with the student's number. For each value of n, $10 \leqslant n \leqslant 57$, the .h file will contain a p sequence and 20 sums obtained by summing some terms of that sequence. The goal is to find out, for each sum, which terms were used to get the sum.

Please place your code in the `subset_sum.c` file. Each group of students should solve as many problems as they can, using the `.h` files corresponding to their student numbers — three students, three `.h` files! You can, and should, **delete** the `.h` files that are of no interest to you (the files for the students not in your group).

Things have been set up to make this easy. For example, for a student with number 100000, the program can be compiled using

```
cc -Wall -O2 -DSTUDENT_H_FILE=\"100000.h\" subset_sum_problem.c
```

so there is no need to hardwire the `.h` file name in the C file. As an alternative, you can edit the `makefile`, replacing the `.h` file name in the line

```
STUDENT_H_FILE=000000.h
```

There exists a special `.h` file, named `000000.h`. It contains a full set of problems. The solutions to these problems are stored is comments after each sum; in each comment, the bits appear in the order $b_1, b_2, \ldots, b_n$. Use these solutions to check the correctness of your own program.

If the Schroeppel and Shamir technique is successfully implemented by a group of students (see next slide), it becomes feasible to break the 64-bit barrier, and to solve problem instances with somewhat larger values of n (up to 64). The program will have to deal with integers of type `unsigned __int128`. For those students, larger sets of problems instances will be available upon request.

Merkle-Hellman cryptosystem (part 4, what has to be done)

Each group should try the following techniques to solve the subset-sum problems:

- Brute force, using a recursive function.
- Clever brute force (called branch and bound), using a recursive function that avoids further recursions when it can be proved that with the current data no solution is possible because the partial sum is either too small or too large.
- Horowitz and Sahni technique (meet-in-the-middle technique). In this technique the p data is split in two nearly equal parts, all possible subset sums of each of the two parts are stored in sorted order in two arrays, and, in a single pass over these sorted arrays, one looks for the solution. In particular,
  1. Let a be the first sorted array, with size n_a, and let b be the second sorted array, with size n_b.
  2. Let i the be a-array index and let j be the b-array index; i starts at 0 and j starts at $n_b - 1$.
  3. The goal is to find i and j such that $a_i + b_j = s$, where s is the desired sum. In each loop iteration of the algorithm either the solution is found or either i is increased if $a_i + b_j < s$ — that is the only way of increasing the sum — or j is decreased if $a_i + b_j > s$ — that is the only way of decreasing the sum.
  4. If one of the indices goes out of bounds, no solution exists.
- Schroeppel and Shamir technique (meet-in-the-middle using less memory). This is similar to the Horowitz and Sahni technique but the two arrays a and b are not kept in memory. Instead, the elements of each one are generated on-the-fly using a min-heap (for the a array) and a max-heap (for the b array). The p data has to be subdivided in four nearly equal parts. This is the more difficult part of the assignment, and should be attempted only by good students. No further details are given here. You are on your own.

Dynamic programming techniques, which appear in numerous sites on the internet, are **not** feasible for almost all of the problems in the `.h` files.

Merkle-Hellman cryptosystem (part 5, the written report)

The written report must have:

- A title page (front page) with the name of the course, the name of the report, date, the numbers and names of the students, and an estimate of the percentage each student contributed to the project. The sum of percentages should be 100%.
- A short introduction describing the problem; the source of any material adapted from the internet must be properly cited.
- A **small** description of the methods used to find the solutions.
- A description of the solutions found; this should include graphs of the execution time of the program as a function of the size of the problem.
- Comments or attempts at explanations of the results found. This can be placed near where the results are presented in the report.
- An appendix with all the code and data (bit vectors) of all solutions. Use a small font.
- **Deliverable: one PDF file. No archives, please!**

Suggestion: to simplify placing the solutions in the report, make your program output them in a format that makes the work of pasting them in the report easy. The same applies to the execution times (both for making tables to to making graphs).