Polynomial Time Algorithm for Easy Knapsack Problem

- Input: $A = \{a_1, \ldots a_n\}$ is super-increasing sequence, S
- Output: TRUE and P – binary array of n elements, $P[i] = 1$ means: a_i belongs to subset of A that sums to S, $P[0] = 0$ otherwise. The algorithm returns FALSE if the subset doesn't exist

for $i \leftarrow n$ **to** 1

    if $S \geq a_i$

        then $P[i] \leftarrow 1$ **and** $S \leftarrow S - a_i$

    else $P[i] \leftarrow 0$

if S != 0

    then return (FALSE – no solution)

else return ($P[1]$, $P[2]$, … $P[n]$).

Example

- Alice Private Key:
  - $A = \{1, 2,$ [illegible] $8\}$, $M = 17$, $W = 7$, $2 \le W < 17$, $(7, 17) = 1$
- Public Key:

$B = \{7 \bmod 17, 14 \bmod 17, 28 \bmod 17, 56 \bmod 17\} = \{7, 14, 11, 5\}$

- Bob Encryption:
  - Plaintext: 1101
  - Ciphertext = 7 + 14 + 5 = 26
- Alice Decryption:
  - $w = 5$ – multiplicative inverse of 7 (mod 17)
  - 5*26 (mod 17) = 11
  - Plaintext: 1101 (11 = 1*1 + 1*2 + 0*4 + 1*8)

Alice
Knapsack Cryptosystem Construction

- Chooses $A = \{a_1, \ldots a_n\}$ super-increasing sequence, A is a private (easy) knapsack
  $a_1 + \ldots + a_n = E$
- Chooses M - the next prime larger than E.
- Chooses W that satisfies $2 \leq W < M$ and $(W, M) = 1$
- Computes Public (hard) knapsack $B = \{b_1, \ldots . b_n\}$, where $b_i = Wa_i \pmod{M}$, $1 \leq i \leq n$
- **Keeps Private Key: A, W, M**
- **Publishes Public key: B**

Bob – Encryption Process

- Binary Plaintext P breaks up into sets of n elements long: $P = \{P_1, \dots P_k\}$
- For each set P_i compute $\sum_{j=1}^{n} P_{ij} b_j = C_i$
- C_i is the ciphertext that corresponds to plaintext P_i
- $C = \{C_1, \dots C_k)$ is ciphertext that corresponds to the plaintext P
- C is sent to Alice

Alice – Decryption Process

- Computes w, the multiplicative inverse of W mod M:
  $wW \equiv 1 \pmod{M}$
- The connection between easy and hard knapsacks:
  $Wa_i = b_i \pmod{M}$ or $wb_i = a_i \pmod{M}$ $1 \le i \le n$
- For each C_i computes: $S_i = wC_i \pmod{M}$

$$S_i = wC_i = w\sum_{j=1}^{n} P_{ij} b_j = \sum_{j=1}^{n} P_{ij} w b_j = \sum_{j=1}^{n} P_{ij} a_j$$

- Plaintext P_i could be found using polynomial time algorithm for easy knapsack