Cosc 412: Cryptography and security
Lecture 5 (5/8/2020)
Complexity, knapsacks, and attacks

Michael Albert
michael.albert@cs.otago.ac.nz

This week

- P, NP, and all that stuff
- Knapsack cryptosystems
- Attacks on knapsacks
- Some RSA attacks
- Other uses for public key encryption

P, NP, and all that stuff

- The *size* of a problem (n) is the number of bits required to represent its input.
- The complexity of algorithms are measured in terms of how they scale with the problem's size.
- A problem can be solved in *polynomial time* (is in $\mathcal{P}$) if there is an algorithm to solve it whose running time is $O(n^c)$ for some constant c.
- A problem can be solved in *non-deterministic polynomial time* (is in $\mathcal{NP}$) if checking its solutions is in $\mathcal{P}$.
- A problem is *NP-complete* if it is in $\mathcal{NP}$ and at least as hard as every other problem in $\mathcal{NP}$ (specifically, if it were shown that the problem was in $\mathcal{P}$, then every problem in $\mathcal{NP}$ would be in $\mathcal{P}$).
- The, literally, million dollar questions: are $\mathcal{P}$ and $\mathcal{NP}$ the same thing?

3-SAT (NP-complete)

INPUT:

- A sequence $x_1, x_2, \ldots x_n$ of binary variables.
- A set of clauses, each containing 3 literals e.g., $x_1 \vee x_2 \vee \neg x_3$.

PROBLEM:

Is there a truth assignment to the variables that makes all the clauses true?

Traveling salesman (NP-complete)

INPUT:

- A sequence x_0, x_1, $\ldots x_{n-1}$ of vertices.
- A function f from pairs or vertices to the positive integers.
- A parameter K

PROBLEM:
Is there a permutation y_0, y_1, $\ldots$, y_{n-1} of the vertices such that

$$f(y_0, y_1) + f(y_1, y_2) + \cdots + f(y_{n-2}, y_{n-1}) + f(y_{n-1}, y_0) \leqslant K?$$

Vertex cover (NP-complete)

INPUT:

- A graph G consisting of:
  - A sequence $v_0, v_1, \ldots v_{n-1}$ of vertices.
  - A set E of *edges*, each being an unordered pair of vertices.
- A parameter K

PROBLEM:
Is there a set of K or fewer vertices such that every edge contains at least one vertex in the set?

Digression (fixed parameter tractability

- In many problems there is a parameter K, which is somewhat independent of the problem size n.
- Or there may be relevant parameters within the problem (e.g., for graphs, maximum degree of a vertex).
- If there is an algorithm whose complexity is $O(f(K)n^c)$ for *any* function f then we say that the problem is *fixed-parameter tractable*
- Problems of this type may be efficiently solvable even for quite large values of n if the parameter K is sufficiently small.

FPT example: vertex cover

- Pick an edge. One of its endpoints must be in any cover. Build a binary tree to depth K. With care $O(2^k n)$ (the 2 can be improved to 1.29).
- Reduce the problem to a small *kernel* as follows:
  - If there is a vertex with more than K neighbours it must belong to any successful solution.
  - Include it, delete the edges it covers, and continue (with parameter $K-1$).
  - If not, every vertex covers at most K edges, so K vertices can cover only K^2 edges so if graph has more than K^2 edges (or more than K^2+K vertices) no solution exists.
  - If all is well so far, then we have at most K^2+K vertices so just check each K element subset by brute force (or as above!)
- *What is known about vertex-cover kernelization*.

The holy grail of public key cryptosystems

Consists of (at least) three parts:

- Find an NP-complete problem for which almost all random instances are hard.
- Build a trap-door function around it that can only be opened by solving a random instance.
- Make sure it's resistant to quantum attacks (just in case).

It's not clear that this is completely achievable – though modern forms of *homomorphic encryption* come close. What follows is a tale of how you can go wrong . . .

The subset sum problem

Input: A collection of positive weights, $w_1, w_2, \ldots, w_n$, and a positive integer S.

Problem: Find a vector $b \in \{0, 1\}^n$ such that

$$b_1 w_1 + b_2 w_2 + \cdots + b_n w_n = S.$$

Note we can also just ask the decision version (does such a vector exist). If we have access to a polynomial-time decider then we can just use it (at most) n times to build a solution.

Putting the decision problem in context

What's wrong with the following algorithm?

- Initialise a boolean array *sums* indexed from 0 to S, with $sums[0] = T$ (and all others F).
- For each w_i scan *sums* and, whenever $sums[j] = T$ set $sums[w_i + j] = T$ (assuming $w_i + j \leq S$)
- If we ever set $sums[S] = T$ we're done. If not, we're done too.

That's $O(nS)$ so polynomial right?

No! The *size* of the problem is n (the number of weights) times the maximum number of bits required to represent a weight, plus the number of bits required to represent S. In that situation S itself is an exponential parameter.

Best known algorithms

- Split the weights into two groups of equal size
- Compute all sums of subsets of weights in each group
- Sort each set of sums
- Scan one list from the bottom and the other from the top, looking for a pair that add up to S
- Number of operations needed proportional to $n2^{n/2}$ and $2^{n/2}$ storage needed

Easy problems

If the weights are *super-increasing*, i.e, for all $1 \le i \le n$

$$w_i > \sum_{j=1}^{i-1} w_j$$

then a greedy approach works.

From easy to hard (Merkle and Hellman)

How can we convert an easy knapsack problem into a hard one?

- Start with a super-increasing sequence of weights u_1 through u_n ($u_1 \simeq 2^n$, $u_n \simeq 2^{2n}$)
- Choose $M > \sum_{i=1}^{n} u_i$ and W with $\gcd(M, W) = 1$
- Compute $v_i = u_i W \pmod{M}$ for $1 \leq i \leq n$
- Let $w_1, \ldots, w_n$ be the v's in sorted order
- Now, if someone doesn't know M and W, knapsacks based on w's look hard, but since you do, they're easy
- If you like, iterate this process a couple of times

Cryptosystem

- Hide your easy weights as above, and announce w
- To encrypt, sender just computes $b \cdot w$
- To decrypt, you undo the modular multiplication to get back to the super-increasing context and work that out
- Much faster than RSA (a couple of orders of magnitude)
- Larger message size (double the number of bits)
- Larger key size

Too good to be true?

- Unfortunately yes
- Clearly leaks one bit of information (the exclusive or of the bits of b corresponding to odd a's)
- Two basic kinds of attacks - one based on elementary arithmetic and one based on more complex lattice reduction techniques
- The main point is that it's enough to find *some* multiplier and modulus that turn a super-increasing system into the announced one – you don't have to get it exactly right

Signatures in RSA

- RSA is quasi-symmetric in that messages encoded with the private key could be decoded using the public key
- This allows a simple signature mechanism
- Bob transmits (with Alice's public key):

$$E(p_{\text{alice}}, \text{“From Bob: } E(s_{\text{bob}}, m)\text{”})$$

- Alice strips the header and decodes the message with Bob's public key
- So long as Bob's private key is private, no one else could have sent the message

Attacks on RSA

- Key point: all attacks are on standards or implementations, none on the mathematics
- Math is easy – software and hardware are hard!

Some attacks

- Timing attacks – when "bad" ciphertext is found, the speed with which this happens can leak what sort of error occurred. An attacker can use this on multiple modifications of a ciphertext to dig out information about it (end of PKCS1 video)
- Using small secret keys (trying to speed up decryption) allows other "math-based" attacks (Second half of "Is RSA one-way video")
- Another timing attack - the time to compute c^d can expose d
- Similarly, power attack - the power used to do it (smartcard) can expose d (via repeated squaring – this is real!)
- Fault attack – an error in decryption can reveal d (middle of "RSA in Practice" video)

Entropy attacks

- What should happen - the keyspace for RSA is so large that there should never be collisions (see “birthday paradox”)
- But keys are generated randomly - what if devices generating keys have “too little entropy” (so that pseudo-random generation is used)?
- Collisions might occur - frequently in one of the two primes, but not the other
- If $N_1 = pq_1$ and $N_2 = pq_2$ then $p = \mathrm{gcd}(N_1, N_2)$ and both channels are now effectively in the clear
- Try a big web crawl (and some clever tricks to pool gcd computations)
- About 0.5% of keys (in some measure) busted