Great Ideas in Computing

University of Toronto CSC196
Winter/Spring 2019

Week 10: November 23-27 (2020)

Announcements

Announcements

- I have posted the start of Assignment 4 (A4). A4 is due December 7.
- The next quiz was scheduled for November 30 but following our poll on Friday, November 20, we have moved the quiz date to Wednesday, December 2.
- Did anyone take the "trolls vs real people" test? If so, what was your score?
- You might find the following article interesting about super infection spreaders and how it relates to the Barabasi and Albert model mention in week 8.
  https://www.wired.com/story/covid-19-vaccine-super-spreaders/

This weeks agenda

Agenda

- We will continue our discussion of complexity theory, NP-completeness and the $P \neq NP$ conjecture.
- We ended the Friday, November 20 class on slide 13 (without drawing too much on slide 14.

  We will begin today by restating the problems given on slide 13 working our way towards the definition of NP-completeness.
- Then we begin complexity based cryptography.

Some examples of decision problems in NP and believed to not be in P

In all of the examples below we always assume some natural way to represent the inputs as strings over some finite alphabet. In particular, integers are represented in say binary or decimal. Polynomial time means time bounded a polynomial $p(n)$ where n is the length of the input string. (I will explain each of the following decision problems as we introduce them.) These problems are decision variants of optimization problems, relations and functions)

- $SAT = \{F | F$ is a propositional formula that is *satisfiable*$\}$
- $PARTITION = \{(a_1, a_2, \ldots, a_n) | \exists S : \sum_{a_i \in S} a_i = \frac{1}{2} \sum_{i=1}^{n} a_i\}$
- $VERTEX\text{-}COLOUR = \{(G, k) | G$ can be vertex coloured with k colours$\}$
- $FACTOR = \{(N, k) | N$ is an integer that has a proper factor $m \leq k\}$

While the theory of NP completeness is formulated in terms of decision problems, we will be able to find certificates for the above problems with respect to natural verification predicates.

Blank page for explaining decision problems on last page

NP completeness

We will see that with the (conjectured) exception of FACTOR, the other decision problems are *NP*-complete, a concept we will now motivate and define.

A decison problem (or any problem) L is *NP*-hard if *every* problem $L' \in NP$ can be "efficicently reduced" to L. There are different notions of how to formalize "efficiently reduced" and we will discuss this shortly.

A problem L is *NP*-complete if it is both in *NP* and *NP* hard.
Here are the immediate consequences of a problem being *NP*-complete.

- If L is *NP* complete, and $L \in P$, then every $L' \in NP$ is in P
- Equivalently, if any $L' \in NP$ is not in P, then every *NP*-complete problem is not in P.
- There are hundreds (and really thousands) of problems that are *NP*-complete and since we *"religously"* believe $P \neq NP$, we believe that none of these complete problems can be decided in polynomial time.

Why the religious belief and co-NP

Why do we believe so strongly that $P \neq NP$. It is simply that many very talented people over literally centuries have tried to efficiently solve problems that are in NP (especially those that are NP-complete) and failed to do so.

Even so, there have been surprises in complexity theory and one still has to keep in mind that $P \neq NP$ is still a conjecture and not a proven result,

Lets now go back to the claim that we probably cannot efficiently prove that a graph G does *not* have a Hamitonian cycle. That is, what is a short certificate verifying that G does not have a HC.

Let $L \subseteq \Sigma^*$ be a language. We define the complement language $\bar{L} = \{x | x \notin L\}$. So, for example, $\bar{L}_{HC} = \{G | G \text{ does not have an HC}\}$
Note: We are omitting a discussion about strings that do not encode say graphs as this is mainly a "detail".

We then define co-NP as the class of languages $\bar{L}$ such that $L \in NP$; that is, the complement of languages in NP.

Returning the concept of reduction

At the heart of *NP* completeness and more generally algorithm analysis is the concept of (efficient) reduction of problems. When we say that problem A "efficiently" reduces to problem B, we can conclude that an efficient algorithm for B will result in an efficient algorithm for A (and equivalently, the contrapositive states that A not efficiently computable implies that B is not efficiently computable).

There are different definitions for what we mean by an efficient reduction and the precise definition matters in terms of what we want to conclude from the reduction.

One major distinction is between a very general type of reduction (which we will just call *poly time reduction* and a more restricted type which we will call *poly time transformation*.

Two types of reductions continued

The general version of reduction $A \leq_{Cook} B$ means that there is a poly time algorithm ALG that can call a subroutine for B and ALG computes A. Here we count each call to the subroutine as 1 step. It is not difficult to see that if $A \leq_{Cook} B$ and B is computable in polynomial time, then A is computable in polynomial time.

The $\leq_{Cook}$ reduction is what Cook used in his seminal 1971 paper.

The more restricted transformation $A \leq_{Karp} B$ means that there is a polynomial time function h (transforming an input instance of A to an input instance of B) such that $x \in A$ if and only if $h(x) \in B$. It is again easy to see that $A \leq_{Karp} B$ and $B \in P$ implies $A \in P$.

Following Cook's paper, Karp provided a list of 21 combinatorial and graph theoretical problems that are NP complete. Karp used the more restrictive $\leq_{Karp}$.

Another conjecture: $NP \neq$ co-NP

FACT: If L is NP-complete then $\bar{L} \in$NP if and only if $NP =$ co-NP

This is another widely believed conjecture again based on the inability of experts to show that $\bar{L} \in NP$ for some NP-complete problem.

As I mentioned before, we believe factoring intergers is not polynomial time computable. In fact, there is a sense in which we believe it is not polynomial time computable "on average" (whereas the basic theory of NP completeness is founded on worst case analysis).

One question on Assignment 4 is to show how a polynomial time algorithm for the decision problem *FACTOR* would allow us to compute the unique prime factorization of an integer.

Perhaps surprisingly, co-*FACTOR* is in NP. That is, given an input (N, k), we can provide a certificate verifying that N does *not* have a proper factor $m \leq k$.

Since co-*FACTOR* is in NP, and we conjecture that $NP \neq$ co-NP, this leads us then to believe that *FACTOR* is in $NP \setminus P$ but *not* NP-complete.

Returning to the two different reductions

As far as I know, there is no proof that the two reductions are different but there is good reason to believe that they are different in general.

- Clearly $\bar{A} \leq_{Cook} A$ for any language A.
- $A \leq_{Karp} B$ and $B \in NP$ implies $A \in NP$.
- Hence our assumption that $NP \neq co - NP$ implies that we *cannot* have $\bar{A} \leq_{Karp} A$ for any NP-complete A.

On the other hand as far as I know all known NP complete problems can be shown ito be complete using transformations $\leq_{Karp}$. So I would usually just says $\leq_{poly}$ and unless stated otherwise take this to mean transformations.

I know of no compelling evidence that general reductions and transformations are different when resticted to the class NP.

NOTE: The general reduction concept makes sense when reducing say a search or optimization problem to a decision problem (and indeed this is what we will be doing next). On the other hand, transformations are only about decision problems (i.e., languages).

Finding a certificate for an NP-complete problem

One might wonder if we can always efficiently find a certificate. In fact, for NP-complete problems we can. Let L be a NP-complete problem. We can prove that for every YES input instance x (where we know that a certificate exists) that a certificate can be computed in polynomial time assuming we can solve the decision problem in polynomial time.

Of course, we do not believe the decision problem can be solved in polynomial time so this is just a claim that it is sufficient to just focus on the decision problem.

As an example, consider SAT and suppose F is satisfiable. That means we can set each propositional variable (to TRUE or FALSE) so that the formula evaluates to TRUE. So how do we find a satsifying truth assignment for F?

Finding a satsifying assignment for a formula F assuming $P = NP$

Once we assume $P = NP$, we would know that the decision problem for SAT is satisfiable. So we would first test if the given formula F is satisfiable. If so, we can construct a saitrsfying assignment one variable at a time. Consider the following example:

$$F = (\bar{x}_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3) \wedge (\bar{x}_3 \vee \bar{x}_1) \equiv (x_1 \rightarrow x_2) \wedge (x_2 \rightarrow x_3) \wedge (x_3 \rightarrow \bar{x}_1)$$

Now since F is satisfiable, there must be some way to set (say) x_1 to either TRUE or FALSE so that the resulting formula still is satisfiable.

If we set x_1 to TRUE, then the resulting formula $F' = F|_{x_1 = TRUE}$ will become FALSE so it must be that x_1 is FALSE in any satisfying assignment.

How would we know that $F' = F|x_1 = TRUE$ is not satisfiable?

Finding a satsifying assignment for a formula F assuming $P = NP$

Once we assume $P = NP$, we would know that the decision problem for SAT is satisfiable. So we would first test if the given formula F is satisfiable. If so, we can construct a saitrsfying assignment one variable at a time. Consider the following example:

$$F = (\bar{x}_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3) \wedge (\bar{x}_3 \vee \bar{x}_1) \equiv (x_1 \rightarrow x_2) \wedge (x_2 \rightarrow x_3) \wedge (x_3 \rightarrow \bar{x}_1)$$

Now since F is satisfiable, there must be some way to set (say) x_1 to either TRUE or FALSE so that the resulting formula still is satisfiable.

If we set x_1 to TRUE, then the resulting formula $F' = F|_{x_1 = TRUE}$ will become FALSE so it must be that x_1 is FALSE in any satisfying assignment.

How would we know that $F' = F|x_1 = TRUE$ is not satisfiable? We would again use the decision procedure SAT applied to F'. We would continue this way to see how to set x_2, x_3. In this example, x_2 can be set TRUE or FALSE and we would just choose one value. In general, a formula can have many satisfying assignments.

End of Wednesday, November 25 class

We ended on slide 13 and will continue from that point. I know some (many?) students may find this to be difficult material as you would not have seen it before. Please ask questions

I do think this material is fundamental to computer science (as a discipline) and computing (in terms of its impact).

Some ideas are great ideas even when we are not that aware of them. I argued that this was the case with respect to Turing's work and the von Neumann model.

The concept of *NP* completeness is something that algorithm designers may or may not think of routinely but at some level of understanding we do need to know that common (say optimization) problems cannot be solved effciciently for all input instances.

I mentioned that there have been many surprises in complexity theory so I again emphasize that a conjecture may guide our thinking but we always have to be aware of what has and has not been proven.

Can randomization help?

We should note that there are many other fundamental questions in complexity theory (in addition to the P vs NP question). One such question is can randomization help.

Consider the following problem: We are *implicitly given* two multivariate polynomials $p(x_1, \ldots, x_n)$ and $q(x_1, \ldots, x_n)$. For example, the polynomials might be the result of a polynomial time computation using the arithmetic operations $+, -, *$. Or p and q might be the determinants of $n \times n$ matrices with entries that are linear functions of the $\{x_i\}$.

The *polynomial equivalence question* whether or not $p \equiv q$ as polynomials; that is, does $p(x_1, \ldots, x_n) = q(x_1, \ldots, x_n)$ for all values of the $\{x_i\}$. Lets say that the x_i are all integers or rationals.
Note that this is the same as asking whether or not $p - q \equiv \mathbf{0}$ where $\mathbf{0}$ is the zero polynomial.

How would you solve the *identically zero* question for a univariate polynomial (again given implicitly)?

Polynomial equivalence problem continued

Fact: A non zero univariate polynomial $p(x)$ of degree d has at most d distinct zeros. This means that if we evaluate $p(x)$ at say $t > d$ random points $r_1, \ldots r_t$, the probability that $p(r_i) = 0$ is at most $\frac{d}{t}$.

Schwartz-Zipple Lemma: This lemma extends the above fact to multivariate polynomials. That is,
If $p(x_1, \ldots, x_n)$ is a non zero polynomial of total degree d (with coeficients in a ring or field F like the integers or rationals) then
$Prob[p(r_1, \ldots r_n) = 0] \leq \frac{d}{|S|}$ when the r_i are chosen randomly in a finite subset $S \subseteq F$.

Polynomial equaivalence and the class *RP*

So to test if p is identically zero, we take $|S|$ sufficienlty large (or do repeated independent trials with say $|S| = 2d$), and see if the evaluation returns a non-zero value. If $p(r_1, \ldots, r_n) = 0$, we will claim that $p \equiv \mathbf{0}$. The error in this claim will be at most $\frac{d}{|S|}$ and we will only make an error if $p \not\equiv \mathbf{0}$.

This is an example of a polynomial time randomized algorithm with 1-sided error (with say error at most $\frac{1}{2}$) and RP is the class of languages that have such an algorithm.

In fact the error can be as big as $1 - \frac{1}{n^k}$ for any fixed k as we can do polynomially many repeated trials to reduce the error probability using the fact that $(1 - 1/t)^t \to \frac{1}{e}$ as $t \to \infty$.

Open question: Is $RP = P$? As a specific example, is the polynomial equivalence problem in P?

RP and BP

Surprisingly, some prominent complexity theorists (but not everyone) believe $P = RP$. More generally, they believe $BPP = P$ where BPP is the class of languages that can be solved by a polynomial time randomized algorithm with 2-sided error (with probability of error at most $\frac{1}{2} - \frac{1}{n^k}$).

Like RP, we can amplify the probability of a correct answer by running a polynomial number of trials and taking the "majority vote" amongst the outcomes of the individual trials.

A langauge in RP can be formulated so that there are many certificates and hence $RP \subseteq NP$.

One final comment about the conjecture $P \neq NP$. While we strongly believe $P \neq NP$, all is not lost if $P \neq NP$. For example, while an optimization problem it can be NP-hard to compute an *optimal solution*, for many NP-hard problems there are efficient *approximately optimal* algorithms. And many natural problems have efficient algorithms when considering restricted classes of (or distributions over) instances that tend to occur naturally.

Complexity based cryptography and Public key encryption

In our discussion of cryptography, I am relying on CSC2426F graduate course notes by Charles Rackoff. See http://www.cs.toronto.edu/ rackoff/2426f20/Cryptonotes.html

I may also be using web page notes by Paul Johnson. See http://pajhome.org.uk/crypt/index.html

What is *complexity based cryptography?*
We are going to explore a counter-intuitive idea: Namely, the ability to use assumptions about what *cannot* be computed efficiently (i.e., negative results) to establish positive results for applications such as pseudo-random number generators, public key cryptography, digital signatures, secret sharing, and more. These applications all fall under the general topic of complexity based cryptography. Our focus will be on pseudo-random number generators (PRNG) and public key cryptography (PKC).

Randomization is necessary

Before we begin, we should note that *randomization* is almost always necessary for cryptography. This is not the first time we have encountered the need for randomization.

When have we used randomization before?

For various problems (say within NP), it seems that randomization is helpful but perhaps not provably so. That is, we do not know if RP and BPP are different from P or NP.

But their are applications where randomization is necessary

Randomization is necessary

Before we begin, we should note that *randomization* is almost always necessary for cryptography. This is not the first time we have encountered the need for randomization.

When have we used randomization before?

For various problems (say within NP), it seems that randomization is helpful but perhaps not provably so. That is, we do not know if RP and BPP are different from P or NP.

But their are applications where randomization is necessary

- Simulating stochastic events
- Hashing
- Differential Privacy

Randomization is necessary

Before we begin, we should note that *randomization* is almost always necessary for cryptography. This is not the first time we have encountered the need for randomization.

When have we used randomization before?

For various problems (say within NP), it seems that randomization is helpful but perhaps not provably so. That is, we do not know if RP and BPP are different from P or NP.

But their are applications where randomization is necessary

- Simulating stochastic events
- Hashing
- Differential Privacy
- And we can add cryoptography to this list

A secure shared secret key session

In this setting, two people called A and B (sometimes referred to as Alice and Bob) have been able to share *secret key* (e.g., a secret string of bits) and will use that secret key to communicate over an insecure channel. This insecure channel can be observed or perhaps even modified by an adversary.

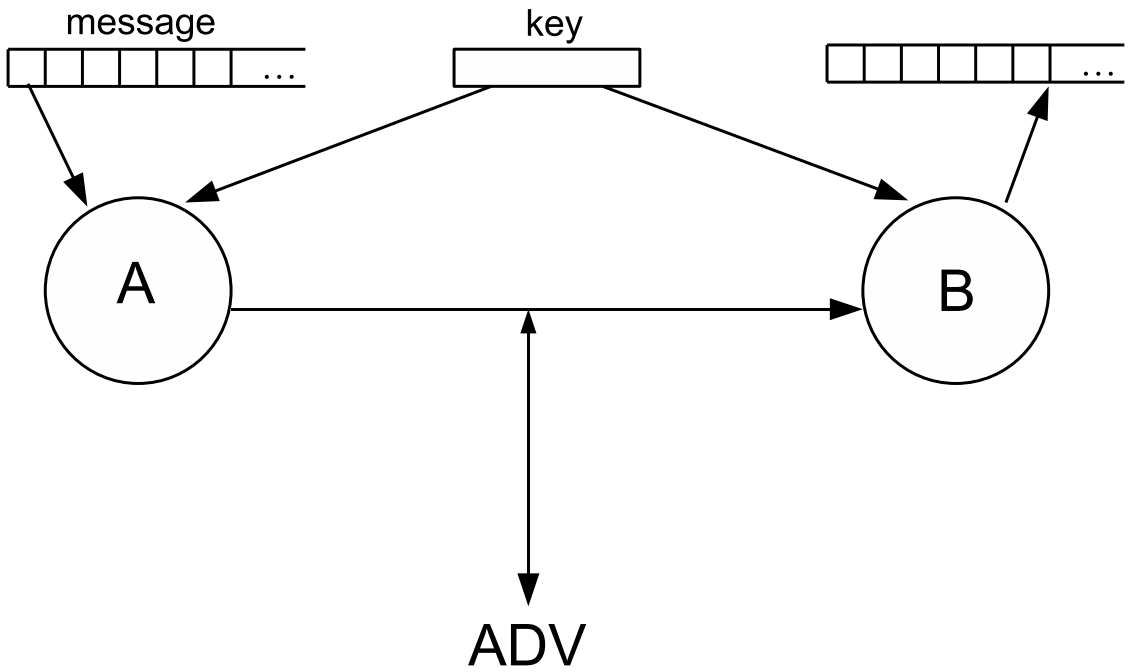

Figure: One-way communication. Figure taken from Rackoff notes

Shared-secret key session continued

An important consideration is how powerful is the adversary.
To do things reasonably carefully, we would probably need a full graduate course on cryptography. It is difficult enough to develop the main ideas even assuming that the adversary can only eavesdrop so lets make that assumption.

In a one-way session, A has an m bit message $M = M_1 M_2 \dots M_m \in \{0,1\}^m$. (For simplicity, we are assuming that the message and the secret key have been reresented as a binary strings but this is not essential.) The message is called the *plain text*.

In the shared secret key setting we are assuming the A and B have agreed upon a secret key n bit key $K = K_1 K_2 \dots K_n \in \{0,1\}^n$.

A will encode his message by a function $ENC : \{0,1\}^m \times \{0,1\}^n \to \{0,1\}^*$. Here we are using the $*$ to suggest that the encoded message length can depend on the plain text message.
Why not a fixed length independent of the message?

Shared-secret key session continued

An important consideration is how powerful is the adversary.
To do things reasonably carefully, we would probably need a full graduate course on cryptography. It is difficult enough to develop the main ideas even assuming that the adversary can only eavesdrop so lets make that assumption.

In a one-way session, A has an m bit message $M = M_1 M_2 \dots M_m \in \{0,1\}^m$. (For simplicity, we are assuming that the message and the secret key have been reresented as a binary strings but this is not essential.) The message is called the *plain text*.

In the shared secret key setting we are assuming the A and B have agreed upon a secret key n bit key $K = K_1 K_2 \dots K_n \in \{0,1\}^n$.

A will encode his message by a function $ENC : \{0,1\}^m \times \{0,1\}^n \to \{0,1\}^*$. Here we are using the $*$ to suggest that the encoded message length can depend on the plain text message. Why not a fixed length independent of the message? The encoded message is called *the cypher text*.

Shared-secret key session continued

B will decode the cypher text by a function
$DEC : \{0,1\}^* \times \{0,1\}^n \to \{0,1\}^m$.

What properties do we want from the exchange?

Shared-secret key session continued

B will decode the cypher text by a function
$DEC : \{0,1\}^* \times \{0,1\}^n \to \{0,1\}^m$.

What properties do we want from the exchange?

- **Privacy:** The adversary should not learn anyything "signifiicant" about the plain text. What does the adversary know in advance about the plain text?
- **Correctness:** B should be able to correctly decode the message. That is $DEC(ENC(M,K),K) = M$ for all M and K.

We might ask what is *perfectly secure session*? In the Rackoff notes #0, there are three equivalent definitions. Let's just use the first one. For a given plain text message M, a uniformly random key K induces a distribution D_M on the cypher texts. The session is perfectly private if the distribution D_M does not depend on M.

Is a perfectly secure session attainable?

End of Friday, November 27 class

We ended at slide 23.

We will finish up our discussion of complexity based cryptography next Friday, December 4.

The second and final quiz will take place, Wednesday, December 2 during the class time.

When is a perfectly secure session attainable?

Fact: A Perfectly secure session is attainable if and only if $|M| \leq |K|$.

When $|M| \leq |K|$, a *one-time pad* provides a single perfect secure session. A one-time pad is defined as follows:

$ENC(M, K) = E_1 E_2 \ldots E_m$ where $E_i = M_i \oplus K_i$ for $1 \leq i \leq m$ and $DEC(E, K) = E_1 \oplus K_1, \ldots, E_m \oplus K_m$.

Note that $\oplus$, the exclusive OR, flips a bit.

Warning: Never use an old key for a new purpose. For example, we cannot securely send two m bit messsages with the same m bit key.

So how are we going to continually generate random private keys (or long keys that can be partitioned into session keys) for different people to communicate? We cannot assume people can get together physically and even so how can they generate truly random strings of bits?

Complexity based assumptions; public key cryptography

The one-time pad does not need any assumptions and an adversary can have unlimited computational power and still cannot gain any information from a one-time pad. But as we noted, a one-time pad is not a very practical solution especially for frequent transactions in e-commerce.

The major application of *public key cryptography* is to enable key exchange. For public key cryptography (and almost all cryprographic applications) we will need complexity assumptions stronger than (but still widely accepted) $P \neq NP$. To make public key systems practical we will also need some sort of trusted public key infrastructre.

We will just discuss one well known public key system, *RSA*, which is based on the assumption that factoring large integers is hard even in some average sense (rather than worse case sense). This is a much stronger assumption than $P = NP$ since $P = NP$ would allow us to factor integers in polynomial time.

The basic idea of public key encryption

Public key encryption was introduced by Diffie and Hellman, and a particular method (RSA) was created by Rivest, Shamir and Adelman.

The basic idea is that in order for Alice (or anyone) to send Bob a message, Bob is going to create two related keys, a public key allowing Alice to send an encrypted mesasage to Bob, and a private key that allows Bob to decrypt Alice's message.

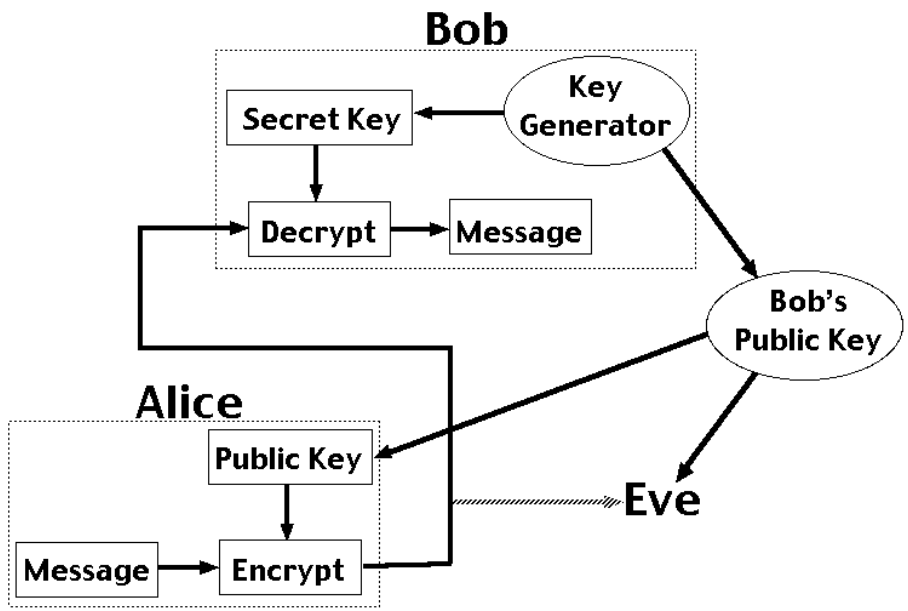

Figure: Diagram of public key encryption. Figure taken from Paul Johnston notes

The RSA method

Bob wants to generate two keys, a public key e, N and a private key d. The claim is that it is hard on average to find d given e and N. Bob chooses $N = p \cdot q$ for two large primes p, q (which for defining "on average" may satisfy some constraint).

Bob will choose the public e such that $gcd(e, \phi(N)) = 1$ where $\phi(N) = \phi(pq) = (p-1)(q-1)$. $\phi(N)$ is called the Euler totient function which is equal to the number integers less than N that are relatively prime to N. $gcd(a, b) = 1$ means that a and b are relatively prime (i.e. have no common proper factors).

Alice encodes a message M by computing M^e mod N.

Hiding some mathematics, BOB can compute a d such that $de = 1$ mod $(p-1)(q-1)$ since Bob knows p and q. But without knowing p, q, finding d becomes computationally difficult.

Hiding some more mathematics, it will follow that $M^{de} = M \pmod{N}$ for any message M. That is, Bob decrypts a cypher text C by the function C^d mod N.

What mathematical facts do we need to know.

The main mathematical facts are :

1. There are sufficiently many prime numbers in any range so one can just randomly try to diffent numbers and test if they are prime.
2. $a^{\phi(N)} = 1 \bmod N$ for any a such that $gcd(a, N) = 1$ As a special case, $a^{p-1} = 1 \bmod p$ for any prime p and a not a multiple of p. So we have $M^{(p-1)(q-1)} = 1 \bmod N$.
3. If $gcd(a, b) = 1$ then there exists s and t such that $sa + tb = 1$. In the RSA algorithm, we can let $a = e$ and $b = (p-1)(q-1)$. Then s will become the d we need for decryption. That is $de + t(p-1)(q-1) = 1$.
4. It follows then that $M^{de} = M^{1-t(p-1)(q-1)} = M \cdot M^{-t(p-1)(q=1)} = M \bmod (p-1)(q-1)$.

What computational facts do we need to know?

1. The extended Euclidean algorithm can efficiently compute an s and t such that $sa + tb = gcd(a, b)$
2. a^k mod N can be computed efficiently for any a, k, N.
3. We can efficiently determine if a number p is prime.

In practice, public keys e are chosen to be reasonably small so that encryption can be made more efficient.

Note that we have been assuming that an adversary EVE (i.e., is just eavesdropping) and not changing messages. That is, EVE just wants to learn the message or something about the message. If EVE could change messages then EVE could pretend to be BOB. So one needs some sort of a public key infrastructure.

Note that if EVE knows that the message M was one a few possibilities, then EVE can try each of the possibilities; that is compute $M^e mod N$ for each possible M to see what message was being sent. So here is where randomness can be used. We can pad or interspers random bits in the plain text M so that the message being sent becomes some one of many random messages M'.

WARNING: Real world cryptography is sophisticated

Complexity based cryptography requires careful consideration of the definitions and what precise assumptions are being made.

Complexity based cryptography has led to many important practical protocols and there are a number of theorems. Fortunatley, many complexity assumptions turn out to be equivalent.
In the Rackoff notes, the following theorem is stated as the fundamental theorem of cryptography. (To make this result precise, one needs precise definitions which we are omitting.)
Theorem: The following are equivalent:

- It is possible to do "computationally secure sdssions"
- There exists pseudo-random generators; that is, create strings that comoutationally look random)
- There exist one way functions f; that is functions such that $f(x)$ is easy to compute but given $f(x)$ it is hard to find a z such that $f(z) = f(x)$
- There exist computationally secure digital signature schemes.