These are lecture notes for the course 2D1441 *Seminars on Theoretical Computer Science* given at KTH, spring 2003. They were written by Jon Larsson (with corrections by Johan Håstad) and based on notes taken on the lectures of the March 18th, 2003.

1 Cryptographic Assumptions

Information theory is the foundation of some forms of cryptography, such as one-time pads and certain forms of multi-party computation. However, most modern cryptography rests on one or more assumptions. Such assumptions are usually in the form of a certain computational problem which is difficult to solve.

The reasons for the current discussion of assumptions is two-fold. Firstly, knowledge of the standard assumptions is definitely part of any cryptographer's general knowledge. Secondly, some of these assumption will be used at various points in the course and hence will be of immediate use to us.

1.1 Computational Intractability

The course will use as its basic principle that polynomial time is feasible while non-polynomial time is not feasible. Many of our algorithms will use randomness and thus in fact we will use probabilistic polynomial time as our basic model of computation.

Many cryptographic problems have the property that cracking them is a problem that lies in the complexity class NP. Thus to be able to do any interesting cryptography we need to assume that $NP \neq P$, or more strongly that $NP \not\subseteq BPP$[1]. However this assumption is not sufficient for at least two reasons.

The first reason is that the only problems that this assumption allows us to conclude are difficult are the NP-complete problems and essentially no problem that we will encounter is NP-complete.

The second reason is that the standard complexity theoretic formulation of the P vs NP problem is a worst case formulation. It only discusses algorithms that always return the correct solution. In particular if $NP \neq P$ then we know that for each polynomial time algorithm trying to solve an NP-complete problem there is some instance on which it fails. We would not be content with a cryptosystem that could be broken 99% of the time and hence we need to discuss *average case* complexity.

There is a theory concerning tuples (L, μ), where L is a language in NP and μ is a probability measure on the inputs, which can be used to formally define average-case polynomial time. It's a rather well-developed theory that even has a number of complete (mutually reducible) problems. However, these problems are again not sufficient for us. They are rather few and rather artificial and in fact they do not share a property that is shared by most problems in cryptography, namely that they are in fact generated in such a way that along with the problem also the solution is generated. For instance, in an encryption scheme the user has the key which would enable an attacker to break the system. For problems that are hard in the average case theory the algorithm generating the hard instances does not have such access to the solutions.

[1] Bounded Probabilistic Polynomial time

The conclusion here is that normal complexity theory does not provide what cryptographers need. Something else is required. Enter the *one-way functions*.

1.2 One-way Functions

One-way functions are used in nearly all modern cryptography. A one-way function $f : X \mapsto Y$ has two basic properties:

1. It is easy to compute: $x \mapsto f(x)$ must run in time polynomial in $|x|$.
2. It is difficult to invert: $f(x) \mapsto x$ must **not** run in time polynomial in $|x|$ or, equivalently, polynomial time inversion succeeds with probability $o(n^{-c})\forall c$.

Looking further at the second condition, we can ask ourselves if we require that f must be injective (1-1). In fact it turns out that the best definition is not to require that a one-way function is invertible and we allow an inverter to come up with any preimage of a given value. Let us formalize this and consider the following procedure.

1. Choose $x \in \{0,1\}^n$ randomly.
2. Compute $y = f(x)$ and give y to the adversary A.
3. A returns z and wins if $f(z) = y$.

A function is one-way iff, for any polynomial time adversary, the probability that the adversary wins is smaller than any inverse polynomial.

To be more precise, for any such A and any constant c there is a length n_0 such that for all $n > n_0$ the probability that A wins the above game is bounded by n^{-c} provided that $n > n_0$.

The property of being a one-way function is sometimes not sufficient to allow a certain cryptographic construction and hence one might look for stronger assumptions by asking for a one-way function with special properties. One subclass of functions that *is* useful is that of the *one-way, length-preserving permutations*. A one-way, length-preserving permutation is a function $f : \{0,1\}^n \mapsto \{0,1\}^n \forall n$. It is both one-way and injective and in general much nicer to work with than a general one-way function.

An even more restricted class of functions is given by the *trapdoor permutations* and these can be seen as a generalization of RSA. A trapdoor permutation is a tuple of functions (G, E, D) (where G is a key generator, E is an encryption function and D a decryption function) satisfying that the computations

$$\begin{aligned} G: &\ \ s \mapsto (e,d) \\ E: &\ \ (p,e) \mapsto c \\ D: &\ \ (c,d) \mapsto p \end{aligned}$$

are all easy, whereas

$$(c,e) \mapsto p$$

is intractable.

1.3 Function-specific Assumptions

In reality we use specific, explicitly defined functions, which often have some additional properties. For example, RSA uses integer factorisation, which is based on multiplication being one-way, and Diffie-Hellman key exchange uses discrete logarithms, which are based on exponentiation being one-way. Once again, the one-way nature of these functions is not something that has been proven, but merely assumed.

1.3.1 RSA and Factorisation

In the RSA cryptosystem we choose two large primes p, q and compute their product n. We then choose e, d such that $ed \equiv 1 \bmod \phi(n)$. Now let P be the plaintext and C the crypto text. Then

$$\begin{aligned} C &\equiv P^e \bmod n \\ P &\equiv C^d \bmod n \end{aligned}$$

It is clear that if we can factor integers efficiently then we can break RSA. It is not difficult to prove that if we can, given e and n, find the decryption exponent d then in fact we can also find the factorisation of n using an additional polynomial time computation. This does not prove that breaking RSA is equivalent to factorisation as it might be possible to break RSA without calculating d. In fact the equivalence of breaking and integer factorisation remains unknown.

At first glance, it may seem that we would want it to be true, as that would imply a greater security. However, it would have its merits if inversion of RSA does not imply factorisation of n. For example, having temporary access to a "black box" RSA inverter would not necessarily make it possible to find the factorisation of n.

1.3.2 Diffie-Hellman and discrete logarithms

The discrete logarithm problem is that of given a prime p, a generator g and a number y to find x such that $g^x \equiv y \bmod p$. The Diffie-Hellman key exchange scheme[2] was the first example of how the assumed intractability of discrete logarithm could be used to create an interesting protocol. Let us describe this protocol. The two parties each choose a number, let us call them a and b, at random. They then agree upon a g and a p, compute $g^a \bmod p$ and $g^b \bmod p$ respectively and send the results to each other. The shared secret k is then equal to $g^{ab} \bmod p$. An eavesdropper will see that values g^a and g^b but might still have difficulties finding k.

It is easy to see that if we can compute discrete logarithms, we can break DH. It is unknown whether the converse is true and it might be possible that this key exchange protocol is insecure without the discrete logarithm problem being easy.

There is a variation of this problem, known as *Decision DH* (or simply *DDH*) in which the question is: given (g^a, g^b, g^c), is it possible to tell whether $c = ab$ or c is generated independendly of a and b. The DDH-assumption says that for any polynomial time algorithm A and constant d, if A is given an

[2] henceforth referred to simply as DH

instance that with probability $\frac{1}{2}$ is generated with $c = ab$ and with probability $\frac{1}{2}$ is generated with a random independent c the probability that A can guess which of the two cases applies is bounded by $\frac{1}{2} + n^{-d}$ for any constant d and sufficiently large n where n is the number of bits in the prime p.

Breaking ordinary DH, which is sometimes called *computation DH*, clearly implies breaking DDH but the converse is not known and DDH might be a stronger assumption.

Decision DH may be solvable unless we choose our g and p carefully, due to rather simple reasons. If we choose g to be a generator mod p then a is even iff g^a is a quadratic residue mod p and this can be checked efficiently by calculating $(g^a)^{\frac{p-1}{2}}$ which is 1 if a is even and -1 if it is odd. If a is even then ab is always even while if a is odd then ab is even only half of the time. The standard way to avoid this problem is to choose $p = 2q + 1$, where q is also prime and use a g that generates only the set of quadratic residues mod p. Such a g can be found as $g = (g')^2$ where g' generates the full group.