Definitions and results for Cryptosystems

Ivan Damgård

October 12, 2004

1 Introduction

This note contains a definition of cryptosystems that covers both conventional and public-key systems. We also define what security of such systems mean, and we cite some results on modes of use for conventional cryptosystems, and on existence of public-key systems with various types of security.

2 Cryptosystems

This section assumes you are familiar with the example cryptosystems from chapter 1 in Stinson. A cryptosystem is a triple (G, E, D):

G, algorithm for generating keys: this algorithm is probabilistic and always outputs a pair of keys (k_e, k_d), k_e to be used for encryption and k_d to be used for decryption. We can think of the keys simply as strings of bits.

We then distinguish two types of cryptosystems:

In a *conventional* or *symmetric* system, we always have $k_e = k_d$. Furthermore, there are two fixed finite sets given: $\mathcal{P}$, the set of plaintexts and $\mathcal{C}$, the set of ciphertexts. In most conventional systems, there is also a fixed set of keys $\mathcal{K}$ given, and G takes no inputs, but simply outputs a key chosen uniformly from $\mathcal{K}$. Furthermore, it is often the case that $\mathcal{P} = \mathcal{C}$. For instance in Shift Cipher, or Caesar substitution on the English alphabet, we always have $\mathcal{P} = \mathcal{C} = Z_{26}$. Moreover, a key is generated by choosing a random element from Z_{26}.

In a *public-key* or *asymmetric* system, k_e is different from k_d, in fact we want that it is a hard computational problem to find k_d, even if k_e is known. Furthermore, G takes a *security parameter* k as input, and outputs the key pair (k_e, k_d) as well as the sets of plaintexts and ciphertexts $\mathcal{P}, \mathcal{C}$. The length of the keys produced as well as the sizes of the sets $\mathcal{P}, \mathcal{C}$ now depend on k, and we require that the key length be polynomial in k. The idea is that we can control the security through the choice of k: the larger we choose k, the harder it will be to break the encryption.

E, **algorithm for encryption:** this algorithm takes as input k_e and $x \in \mathcal{P}$ and produces as output $E_{k_e}(x) \in \mathcal{C}$. Note that E may be probabilistic, that is, even though we fix x and k_e, many different ciphertexts may be produced as output from E, as a result of random choices made during the encryption process. In other words, the ciphertext will have a probability distribution that is determined from x and k_e, typically uniform in some subset of the ciphertexts. For instance, consider a cipher, where $\mathcal{P}$ is the ordinary alphabet Z_{26}, whereas $\mathcal{C}$ is Z_{100}, i.e., a much larger set than $\mathcal{P}$. The key then consists of information that splits $\mathcal{C}$ into 26 subsets $A_0, A_2, ..A_{25}$, and to encrypt letter number i, we choose a random element in A_i[1].

D, **algorithm for decryption:** this algorithm takes as input $k_d, y \in \mathcal{C}$ and produces as output $D_{k_d}(y) \in \mathcal{P}$. It is allowed to be probabilistic, but is in most cases deterministic.

For a cryptosystem, we always require that for any pair of keys (k_e, k_d) output by G, correct decryption is possible, i.e. it holds that for any $x \in \mathcal{P}$, $x = D_{k_d}(E_{k_e}(x))$.

All this of course says nothing about security. According to this definition, a system that "encrypts" by sending x to itself is a cryptosystem. Indeed it is, but of course not a secure one!

[1]This is called homophonic substitution, and the original idea historically was to choose larger subsets for encryptions of more common letters, in order to make the frequency distribution of ciphertext letters more flat

3 Attacks on cryptosystems

An attacker against a system in real life may follow a number of different strategies. The easiest option probably is to eavesdrop encrypted communication. But more advanced attackers may be able to guess (or get hold of) part of the plaintext that is being sent and compare this to the ciphertext that is seen. An even more harmful strategy is if the attacker can fool the sender into sending a particular message (chosen by the attacker), for which the attacker can then later see the ciphertext. Conversely, the attacker may be able to fool the receiver into decrypting a particular ciphertext for him.

To model these forms of attacks in a formal definition, we do the following: we think of the adversary as a probabilistic algorithm A. As input, A gets all the data that we know the adversary will always be able to get, such as perhaps the distribution of plaintexts being sent (say, he knows which natural language plaintexts are in), or in case of a public-key system, he knows the public key. Then, to model the data the adversary obtains during the attack, we give A access to an *oracle* O. At any time, A may call the oracle, and obtain an answer. Depending on which type of question we allow the oracle to answer, we can model the different strategies described before:

Ciphertext Only Attack: Some plaintext distribution D is given as input to A, and each time A calls the oracle, it will return $E_{k_e}(x)$, where x is chosen according to D, and k_e is produced by G (but fixed for the duration of the attack).

Known Plaintext Attack: Some plaintext distribution D is given as input to A, and each time A calls the oracle, it will return $x, E_{k_e}(x)$, where x is chosen according to D, and k_e is produced by G (but fixed for the duration of the attack).

Chosen Plaintext Attack: A can call the oracle giving it any $x \in \mathcal{P}$ as input. The oracle returns $E_{k_e}(x)$, where k_e is produced by G (but fixed for the duration of the attack).

Chosen Ciphertext Attack: A can call the oracle giving it any $y \in \mathcal{C}$ as input. The oracle returns $D_{k_d}(y)$, where k_d is produced by G (but fixed for the duration of the attack).

When A stops, it outputs some result. In the worst case (for us), this result is the secret decryption key, but we can (and indeed we must) also

consider adversaries with much less ambitious goals. More on this in the following sections. To get an initial understanding of the difference between types of attacks, consider the Shift Cipher, where the same key is being used to encrypt large pieces of text. Breaking this cipher under a ciphertext only attack is not hard, but requires a little work: one has to do a frequency analysis to determine likely candidates for the encryption of the most common letter, these candidates each suggest a possibility for the key, and one can then try to decrypt a bit of text using each candidate to find the right possibility. The same cipher is even easier to break under a known plaintext attack: just knowing a single plaintext letter and corresponding ciphertext letter immediately reveals the key. A chosen plaintext attack is in general stronger, but in this case it is not more useful to the attacker than a known plaintext attack. Try looking in the same way at attacks on the general substitution cipher. Is it also the case here that a chosen plaintext attack is not worse than a known plaintext attack?

4 Security of Conventional Encryption

This section assumes that you are familiar with either the DES or the AES cryptosystem.

4.1 Deterministic Systems, or Pseudorandom Functions

For simplicity, we will in this section only look at systems where $\mathcal{C}$=$\mathcal{P}$. As an example of such a deterministic system, you may think of DES: we have $\mathcal{P} = \mathcal{C} = \{0,1\}^{64}$, keys are chosen uniformly in $\{0,1\}^{56}$, and as soon as we have specified the key and the plaintext, the ciphertext is uniquely determined. Put another way, each key specifies a function from $\mathcal{P}$ to $\mathcal{C}$ (in fact a permutation), and hence systems of this kind are also called *Function Families*.

Any deterministic system has limits on the kind of security it can provide: if I send the same message today and tomorrow, and I'm using the same key, then I will send the same ciphertext twice, and an adversary listening in can conclude that indeed I sent the same message twice. Even such a limited piece of information can be useful to an enemy, and ideally we would like to hide even such relations between messages sent at different times. So

deterministic systems/funtion families cannot be the final answer to good encryption. Nevertheless, it is useful to talk about their quality because they can be used as building blocks in systems with better security, as we shall see.

So what characterizes a strong deterministic cryptosystem? One natural property that comes to mind is that even if the adversary knows the plaintext for a given ciphertext, the ciphertext appears to be randomly chosen nonsense, with no obvious relation to the plaintext - as long as the key is unknown. To be more concrete, think of DES as an example, and let's assume we give the adversary a chosen message attack. This means that for some fixed (but random) key k, the adversary gets to choose $x_1, x_2, ..$ and gets to see $y_1 = DES_k(x_1), y_2 = DES_k(x_2), ...$ Now, according to the intuition we just outlined, we hope that this appears to the adversary as if $y_1, y_2, ..$ were completely randomly chosen as the images of $x_1, x_2, ...$

Of course, choosing a random DES key, and using it to map x's to y's is not the same as choosing a random function. You can get at most 2^{56} different mappings this way, whereas the total number of functions from $\{0,1\}^{64}$ to itself is astronomically larger: $2^{64 \cdot 2^{64}}$. But it might still be the case that to an attacker with limited resources, DES encryption with a random key *appears to* behave like a random function, we say that the set of DES encryption functions is a *pseudorandom* function family

To define this concept precisely, we consider an adversary A which is placed in one of the following two scenarios, and is asked to guess which one he is in (so he outputs his guess as one bit):

World 0 (the ideal world): A gets an oracle which initially chooses a random mapping R from $\mathcal{P}$ to $\mathcal{C}$ (uniformly among all such mappings), and then on input a plaintext x answers with $R(x)$.

World 1 (the real world): The adversary gets a normal chosen message attack: an oracle which on input a plaintext x answers with $E_{k_e}(x)$, where k_e is produced by G, but fixed in the entire attack.

If we want to formalize the two scenarios completely, we should think of A as being a probabilistic Turing machine. But this is not necessary to understand the concepts. It is sufficient to think of A as being a computer running some program that the adversary has designed. The only two special things about it is that it can ask questions to the oracle and it can make random choices.

Let $p(A, i)$ be the probability that A outputs 1 in world i. These probabilities are taken over A's random choices, and also over the choices made by the oracle in the two cases. The *advantage* of A is defined to be

$$Adv_A = |p(A, 0) - p(A, 1)|$$

DEFINITION 1 *We say that the system (G, E, D) forms a (t, q, ϵ)-secure* pseudorandom function family*, if any adversary that runs in time t, and makes at most q calls to the oracle, has advantage at most ϵ.*

The running time of A is of course well defined if he is a Turing machine. Equivalently (up to constant factors) and closer to reality, one may think of the time as being the number of basic CPU instructions you need in order to execute the adversary's program. Intuitively, you can think of the first two parameters in the definition (t, q) as a measure of how powerful the adversary is, and ϵ as a measure of how successful his attack is. So the if a cryptosystem satisfies the definition, this basically says that as long as the adversary is not too powerful, there is a limit to how successful he can be.

Taking DES as an example, for which values of t, q and ϵ could we hope that DES satisfies this definition? First thing to notice is that DES with a fixed key is not just a function on $\mathcal{P}$, it is a *permutation*. So no matter how many inputs and correpsonding outputs the adversary knows, he will never see a case where two inputs form a *collision*, i.e. they are mapped to the same output. By contrast, a random function will tend to have such collisions, indeed, if the adversary asks for outputs resulting from about $\sqrt{2^{64}} = 2^{32}$ distinct inputs, then there is significant probability that collisions will occur. This follows by the well-known "birthday paradox". So we cannot hope for security as we defined it unless the adversary asks well under 2^{32} quiries. In fact, if you ask q quiries, the probability of a collision in the ideal world will be approximately q^2 divided by the number of blocks. So it will certainly not be more than 2^{-20} in case of DES and 2^{20} quiries. Moreover, the adversary can always first ask 1 or 2 quiries and then search through all possible keys until one is found that matches the data he has. So we also cannot hope for security if the adversary has time comparable to the number of possible keys, 2^{56}. In fact, if he is able to test x keys, he will find the right one with probability $x/2^{56}$, so the advantage cannot be larger than that for any attack that has time enough to test x keys. Even if we are optimistic on behalf of the adversary, he will certainly need at least one CPU instruction per round of DES, so in time 2^{40}, he can test at most $2^{40}/16 = 2^{32}$ keys.

Taking this into account, it seems that $(2^{40}, 2^{20}, 2^{-20})$-security is reasonable to conjecture. Given the current lack of lower bounds in complexity theory, this can be nothing but a conjecture. Nevertheless, some assumption of this type is needed to have security at all. If we had looked instead at AES, which has block- and keylength 128 bits, then much better parameters can be expected, say $(2^{80}, 2^{50}, 2^{-30})$, by the same arguments as above, and allowing some security margin for attacks that may do better than simple search for the key.

The general real/ideal world approach to a definition we have used may not seem like the most natural one at first sight. Why don't we just say that it should be difficult to find the secret key? or that it should be difficult to find any partial information about the plaintext once it has been encrypted? Indeed many such sensible definitions could be proposed. In fact the list of variants is almost infinite. The nice thing about the real-or-ideal-world definition is that it is strong enough such that if a cryptosystem satisfies it, then it also automatically satisfies *any* of the variants we just sketched. The reason is that the ideal world is constructed such that any type of non-trivial "break" is clearly impossible. And then if the adversary cannot even tell if he is attacking the real or the ideal system, he certainly can't succeed in the real world either, for any reasonbale definition of what "succeeding" means.

4.2 Probabilistic Systems

The approach to defining secure symmetric encryption in the probabilistic case is similar to what we did before: we will compare it to an "ideal situation", where it is obvious that an attacker will get nowhere. If an attacker A cannot tell whether he is attacking the real system (G, E, D) or the ideal one, then we say the system is secure against A.

However, a probabilistic cryptosystem can acheive more security than a deterministic one, in fact, we will require that the adversary cannot tell the difference between a real encryption of a message x he chooses, and a completely random ciphertext chosen with no relation to x.

Clearly, in a world where random encryptions are being sent in place of encryptions of the actual messages, the adversary has no chance of figuring out any interesting information at all. For instance, he cannot tell if we send the same message twice. So if he cannot tell the real world from the ideal one, then we have the best possible security.

To model this, we consider again adversary A which is placed in one of

the following two scenarios, and is asked to guess which one he is in (so he outputs his guess as one bit):

World 0 (the ideal world): A gets an oracle which on input a plaintext x answers with $E_{k_e}(r)$, where r is randomly chosen in $\mathcal{P}$ each time the oracle is called, and k_e is produced by G, but fixed in the entire attack.

World 1 (the real world): A gets a normal chosen message attack: an oracle which on input a plaintext x answers with $E_{k_e}(x)$, where k_e is produced by G, but fixed in the entire attack.

As before, we define $p(A, i)$ to be the probability that A outputs 1 in world i, and the *advantage* of A to be

$$Adv_A = |p(A, 0) - p(A, 1)|$$

DEFINITION 2 *We say that the system (G, E, D) is (t, q, μ, ϵ)-secure, if any adversary that runs in time t, and makes at most q calls to the oracle, with plaintexts consisting of a total of μ bits, has advantage at most ϵ.*

Probabilities and running times can be fully formalized in the same as we explained in the previous section for pseudorandom functions.

The reason for the parameter μ, which we did not have in the deterministic case, is that some of the systems we will look at in the following can handle plaintexts of many different lengths. Thus, if we want to measure how much information the adversary has obtained from the oracle, it is not enough to count the number of calls, we must also look at how many bits the adversary asked to have encrypted.

5 Good Symmetric Encryption from Deterministic Schemes

Several standards give schemes by which systems like DES can be expanded to systems which can first handle plaintexts of any length, and second can be probabilistic, with the security advantages this can imply. With the machinery developed above, it is possible to prove that several of these mechanisms really work.

5.1 CBC Encryption

Let a deterministic cryptosystem (G', E', D') be given, where $\mathcal{P}, \mathcal{C}$ are fixed, and $\mathcal{P} = \mathcal{C} = \{0,1\}^k$ for some k.

CBC encryption defines a way to make a new cryptosystem (G, E, D), where first $G' = G$. The plaintext set for the new system will be all strings of length divisible by k. We make this restriction only for simplicity, a small modification of the construction will allow handling strings of any length. To do encryption we choose a random k-bit string y_0, and we split the input x into k-bit strings $x_1, ..., x_t$. Then we define for $i > 0$ that $y_i = E_{k_e}(y_{i-1} \oplus x_i)$. The output ciphertext is $y_0, y_1, ..., y_t$. Decryption is straightforward.

It is now possible to show the following [3]:

THEOREM 1 *Suppose (G', E', D') is (t', q', ϵ')-secure. Then CBC encryption based on this system is (t, q, μ, ϵ)-secure for any q, and for*

$$\epsilon = \epsilon' + \frac{2\mu^2}{2^k k^2}$$

provided that

$$t \leq t' - c\mu, \quad \mu \leq q'k$$

for some constant $c \geq 0$.

Intuitively, what this results says is the following: as long as CBC encryption using (G', E', D') is attacked by an adversary who is no more powerful than what (G', E', D') can handle, the success will be no better than $\epsilon = \epsilon' + \frac{2\mu^2}{2^k k^2}$.

So what the result basically talks about is the difference between ϵ and ϵ'. Let's call a string of k bits a *block*. This is the amount of text the original system can handle in one go. Then we see that the difference between ϵ and ϵ' is basically the square of the number of blocks we encrypt, divided by 2^k. In other words, if we always CBC encrypt much less than $2^{k/2}$ blocks before we change the key, then the advantage with which CBC mode can be broken is essentially the same as the advantage with which the original system can be distinguished from a random function.

Note that although the results says that ϵ may be greater than ϵ', this does NOT mean that CBC mode is less secure than the block cipher itself. We have to remember that the two types of security involved are completely different. What the result really says is that if the original system is secure in

a weak (pseudorandom function) sense, then CBC mode based on this system is secure in a much stronger sense, essentially with the same advantage, as long as we do not encrypt too much data.

Let us give some intuition as to why this result should be true: Let Game 0 be the ideal world as defined above, and let Game 1 be the real world. Then we introduce a new *Game 0.5*, which we think of as being somewhere "in between" Game 0 and Game 1. In this game, the Oracle chooses a random function R from k bits to k bits, and on input some message m from the adversary, the oracle "encrypts" m using CBC mode, but where the block cipher encryption function is replaced by R. We define $p(A, 0.5)$ as the probability that A outputs 1 when playing Game 0.5. Now since the only difference between Games 0.5 and 1 is that $R()$ is replaced by $E_K()$, we must have that $|p(A,1) - p(A,0.5)| \leq \epsilon'$. If this was not the case, A could be used to distinguish between $E_k()$ and a random function with advantage greater than ϵ', contradicting our assumption about the block cipher.

Then observe that Game 0 is completely equivalent to a game where the Oracle on input an n-block message always returns a randomly chosen sequence of $n+1$ blocks. We can then argue that Game 0.5 with high probability does exactly this (and so it also hard to tell Games 0.5 and 0 apart). Define the event BAD as follows: BAD occurs, if at any point during Game 0.5, the function R receives an input that it has received before in this game. Now, if BAD does not occur, since R is a random function, all outputs generated by R will be random blocks, chosen independently of anything else - which means that the output we generate is completely random, exactly as in Game 0. Hence, A's only chance of telling Games 0 and 0.5 apart is if BAD occurs, i.e., $p(A,0) - p(A,0.5)| \leq Pr(BAD)$. This, together with $|p(A,1) - p(A,0.5)| \leq \epsilon'$ from above implies $p(A,0) - p(A,1)| \leq \epsilon' + Pr(BAD)$.

So how large is $Pr(BAD)$? This event occurs only if any of the random blocks we feed into R happen to be the same. The well-known argument for the birthday paradox says that the probability of this happening grows, roughly speaking, as the square of the number of blocks we have, divided by the total number of blocks, in other words, $Pr(BAD)$ is some constant times $(\frac{\mu}{k})^2 \frac{1}{2^k}$. A more careful analysis leads to exactly the bound stated in the theorem.

What happens if we do encrypt $2^{k/2}$ blocks? The result above does not say that anything really bad happens: the bound on ϵ becomes useless, but this does not mean that any attack exists. However, it turns out that CBC encryption does leak information in this case. The reason is the following:

because we have a total of 2^k possible blocks, in a set of $2^{k/2}$ random blocks there will be two that are equal with significant probability. So it happens with quite large probability that two ciphertext blocks y_i, y_j, where $i, j > 0$ are equal (they need not come from the same plaintext). Let x_i, x_j be the corresponding plaintext blocks. Then it follows that

$$E_{k_e}(y_{i-1} \oplus x_i) = y_i = y_j = E_{k_e}(y_{j-1} \oplus x_j)$$

and consequently that $y_{i-1} \oplus y_{j-1} = x_i \oplus x_j$. So from the ciphertext we can compute at least some non-trivial information about the plaintext. In a sense, the above theorem says that if we assume the underlying block cipher is secure, then the attack we just outlined is the best possible against CBC encryption.

For DES, this means that one should not encrypt more than 2^{32} blocks of data under CBC with the same key. This is not too bad a restriction: it corresponds to about 1000 Gbyte (!). Nevertheless, it is an important observation that the blocksize of a cryptosystem influences the security of the CBC construction in this way.

Let us look at a numeric example. Suppose we are willing to believe that DES is a $(2^{40}, 2^{20}, 2^{-20})$-secure system. Then the parameters for DES based CBC will be $2^{40} - c2^{26}, \mu = 2^{26}, \epsilon = 2^{-20} + 2^{-24}$. Of course we do not know how to rigorously prove that DES has the parameters we assumed - but they are not unreasonable, given the known attacks on DES. If we were to use two-key triple DES or AES instead as the basic system, much better parameter values could be reasonably assumed, i.e. larger t, μ and smaller ϵ.

5.2 XOR Mode

This is a construction of the same form as CBC encryption, so we use the same notation as before. The only difference is that the encryption function is defined as follows:

We choose a random k-bit string y_0, and we split the input x into k-bit strings $x_1, ..., x_t$ (we assume for simplicity that the length of x is always a multiple of k). Then we define for $i > 0$ that $y_i = E_{k_e}(y_0 + i) \oplus x_i$. The output ciphertext is $y_0, y_1, ..., y_t$. Here $y_0 + i$ means that you write i in binary and do the addition modulo 2^k. Decryption should be straightforward and is left to the reader.

Just as before there is a result linking the security of XOR encryption to that of the original scheme [3]:

THEOREM 2 *Suppose* (G', E', D') *is* (t', q', ϵ')*-secure. Then XOR encryption based on this system is* (t, q, μ, ϵ)*-secure for any* q*, and where for some constant* $c \geq 0$*:*

$$t \leq t' - 2c\mu \quad \mu \leq q'k \quad \epsilon = 2\epsilon' + \frac{q\mu}{2^k k}$$

6 Security of Public Key Encryption

6.1 Deterministic Systems

Let us take the basic RSA system as an example, and check that it matches the general definition of cryptosystems from the first section: the algorithm G for generating keys is simply whatever algorithm we have for selecting a modulus n, and public and private exponents e, d. In this case the algorithm does take a so called security parameter k as input, which is used to determine the bit length of the modulus produced.

The public encryption key is $k_e = (n, e)$, while the secret decryption key is $k_d = (n, d)$ The set of plaintexts and the set of ciphertexts are the equal, namely $\mathcal{P} = \mathcal{C} = Z_n$. And finally, the encryption operation is $E_{k_e}(x) = x^e \bmod n$, while $D_{k_d}(y) = y^d \bmod n$.

As we see, this system does deterministic encryption, whenever we fix k_e and x, the ciphertext y is uniquely determined. What we actually have here is an infinite family of functions, namely the family consisting of all RSA encryption functions, one for each possible public RSA key. The family is infinite because there is - in principle - no limit on the size of public key we can choose. In, fact any deterministic public-key system defines a family of functions in this way.

What properties can we hope to get from such a family of encryption functions? Well, at least the encryption mappings should be efficiently computable, given the public key, whereas the decryption functions should be hard to compute if you know only the public key, but easy if you know the secret trapdoor: the decryption key. Such a family is also called a family of *trapdoor one-way functions*, a concept first proposed by Diffie and Hellman in their famous "New directions in Cryptography" paper. Of course, since we have an infinite family of functions, we should be careful about what we mean by "easy" and "hard" in this context. Roughly speaking, we want that if we choose long enough keys, then it is still reasonably efficient to compute the encryption function, but it becomes infeasible to compute the decryption

function knowing only the public key.

Here is the standard complexity theory inspired way to formalize these requirements.

DEFINITION 3 *We will say that the system* (G, E, D) *forms* a family of trapdoor one-way functions *if the following is satisfied:*

- *The algorithms* (G, E, D) *defining the system all run in time polynomial in the security parameter* k*.*
- *Let any probabilistic polynomial time algorithm* A *be given. Consider the following experiment: we run* G *on input* k*, let* $(k_e, k_d, \mathcal{P}, \mathcal{C})$ *be the output. Then we select* x *at random in the set of plaintexts* $\mathcal{P}$ *and finally we run* A *on input* $k_e, E_{k_e}(x)$*, i.e. we give* A *the public key and the encryption of a random plaintext. Let* $p(A, k)$ *be the probability that* A *outputs* x*. Then we require that for any* A*,* $p(A, k)$ *is* negligible *in* k*.*

  A probability $\epsilon(k)$ *that depends on* k *is negligible if it holds that for any polynomial* f*, we have* $\epsilon(k) \leq 1/f(k)$ *for all large enough* k*. In other words, asymptotically in* k*, it vanishes to zero very quickly. This is a complexity theoretic way to say that "for all practical purposes", the probability might as well have been zero.*

For the case of RSA, in order to build any security on this system, we have to at least assume that it satisfies the above definition. This is the standard

RSA assumption: the basic RSA algorithm (including the standard method for generating keys) defines a family of trapdoor one-way permutations.

Applying a one-way trapdoor function directly to encrypt a messages gives only very limited security, and is not something one would really use for encryption. One problem is the following: suppose our worst enemy knows that tomorrow we will send one of two messages x_0, x_1. Then he can just use the public key and apply the encryption function to both messages and store the two ciphertexts he obtains. Now he just waits and watches which of the two show up on the communication line tomorrow. Then he knows what we sent.

Furthermore, even if it is hard to compute x from $E_{k_e}(x)$, this does not guarantee that an adversary cannot compute part of x. Even if, say, half the

bits of $x$are easy to compute from $Ek_e(x)$, the system might still satisfy the definition above. Of course this is not a satisfactory security guatantee.

The solution to all this turns out to be probabilistic encryption.

6.2 Security of Probabilistic Systems: Semantic Security

We will first look at security against a passive attack, i.e. one where the adversary just observes ciphertext, and then tries to get information on some of the hidden plaintext. The following definition is known as *semantic security*, it has become standard in the area, and turns out to imply any other reasonable definition.

The idea is as follows: I give you the public key, you choose any message you want, and I give you an encryption of either your message, or a complete random message. If you cannot efficiently guess which kind of ciphertext I gave you, we say the system is secure. So this definition is designed to capture the idea that an encryption should tell you nothing at all about the message: for all you care, encryptions might contain only random garbage. This is basically the same idea as in the corresponding definition for conventional (symmetric) encryption.

A bit more formally, Let (G, E, D) be the system in question. Then we consider the following games that the adversary plays with an oracle. They are very similar to the corresponding ones we considered for conventional encryption. The only real difference is that since this is public-key crypto, the adversary should know the public key like anyone else, this also means he can by himself make as many encryptions as he likes.

World 0 (the ideal world): Input to both adversary A and oracle O is the security parameter k. The oracle runs $G(k)$ to get $k_e, k_d, \mathcal{P}, \mathcal{C}$ and gives $k_e, \mathcal{P}, \mathcal{C}$ to A. A computes a plaintext $x \in \mathcal{P}$ and gives it to O. The oracle responds with $E_{k_e}(r)$, where r is randomly chosen in $\mathcal{P}$ of the same length as x. Finally A outputs a bit b.

World 1 (the real world): Input to both adversary A and oracle O is the security parameter k. The oracle runs $G(k)$ to get $k_e, k_d, \mathcal{P}, \mathcal{C}$ and gives $k_e, \mathcal{P}, \mathcal{C}$ to A. A computes a plaintext $x \in \mathcal{P}$ and gives it to O. The oracle responds with $E_{k_e}(x)$. Finally A outputs a bit b.

We define $p_{A,i}(k)$ to be the probability that A outputs 1 in world i on input k. and the *advantage* of A to be

$$Adv_A(k) = |p_{A,0}(k) - p_{A,1}(k)|$$

DEFINITION 4 *We say that* (G, E, D) *is* semantically secure*, if for all probabilistic polynomial time adversaries* A*, it holds that* $Adv_A(k)$ *is negligible in* k*.*

In more human language: the adversary may be able to see from a ciphertext how long the plaintext is, but other than that, no efficient adversary can tell meaningful encryptions from random encryptions. So intuitively, this means that ciphertexts reveal nothing useful information to the adversary.

Clearly, no deterministic system can be semantically secure: as we saw in the previous section, if the system is deterministic, then the adversary can just encrypt his message by himself and compare the results to what he gets from the oracle.

However, it is possible to use deterministic systems to build new ones that do have this type of security. Let us take RSA as an example. Let (G, D, E) be the following public-key system: G simply generates a pair of RSA keys (n, e), (n, d) in the usual way. However, the set of messages is just $\{0, 1\}$ whereas the set of ciphertexts is Z_n^*. The encryption algorithm E will encrypt a bit b by choosing a random number $x_b \in Z_n^*$ such that the least significant bit of x_b is b. The ciphertext is now $c = x_b^e \bmod n$. Decryption is straightforward: reconstruct x_b using the secret RSA key and extract the least significant bit.

It is not too hard to see that if one can always compute the least significant bit of x from $x^e \bmod n$, then one can invert the RSA encryption function. However, something much stronger can also be shown: if there is a polynomial time algorithm that guesses the least significant bit of x (based on $x^e \bmod n$) with just a slight (non-negligible) advantage over a 50% chance, then RSA encryption can be inverted in polynomial time. From this follows immediately [1]:

THEOREM 3 *Under the RSA assumption,* (G, E, D) *as above is a secure probabilistic cryptosystem.*

This system is of course not very efficient: a one bit message is expanded to an encryption that takes up an entire number mod n. However, there are ways to make this be much more efficient.

In practice, the most common use of RSA is to encrypt keys for conventional encryption. Such keys are usually much shorter than an RSA modulus, so what one does is often to place the key in the least significant end of an RSA block and pad with random bits to the right length. This is exactly what we did above, with the only exception that now we put more than 1 bit into a block. But in fact the Theorem above can be extended to show that it is secure to put $O(\log k)$ bits in a block, so this idea does have some theoretical justification.

It turns out that RSA is not special w.r.t. the possibilities of building really secure systems. In general it can be shown that:

THEOREM 4 *If a family of one-way trapdoor permutations exist, then there exists a semantically secure probabilistic public-key system.*

While the proof of this is technical, the construction itself is quite simple: let f be the given one-way trapdoor function, for simplicity assume it maps k-bit strings to k-bit strings. Then to encrypt a bit b, we choose two k-bit strings x, r at random and send

$$f(x),\ r,\ (r \cdot x) \oplus b,$$

where $r \cdot x$ means the inner product of $r = r_1, ..., r_k$ and $x = x_1, ..., x_k$, i.e., $r \cdot x = \oplus_{i=1}^{k} r_i \wedge x_i$.

6.2.1 Chosen Ciphertext Security

Everything we said so far has been about security w.r.t. a passive adversary, i.e., and adversary that simply looks at the public key and some ciphertext and does his best to figure out what the plaintext was. What happens if we give the adversary a stronger attack? as mentioned, a chosen plaintext attack of course does not add to his power: he has the public key and can encrypt as much as he likes. But a chosen ciphertext attack might certainly help him.

To define chosen ciphertext security, we reuse the same idea as before, but in addition, the oracle will now kindly decrypt ciphertext for the adversary:

World 0 (the ideal world): Input to both adversary A and oracle O is the security parameter k. The oracle runs $G(k)$ to get $k_e, k_d, \mathcal{P}, \mathcal{C}$ and gives $k_e, \mathcal{P}, \mathcal{C}$ to A. A may now submit an input string y to O, and O

will return $D_{k_d}(y)$ to A. This is repeated as many time as A wants. Then A computes a plaintext $x \in \mathcal{P}$ and gives it to O. The oracle responds with $y_0 = E_{k_e}(r)$, where r is randomly chosen in $\mathcal{P}$ of the same length as x. A may now again submit an input string y to O, the only restriction is that y must be different from y_0. O will return $D_{k_d}(y)$ to A. This is repeated as many time as A wants.

Finally A outputs a bit b.

World 1 (the real world): Input to both adversary A and oracle O is the security parameter k. The oracle runs $G(k)$ to get $k_e, k_d, \mathcal{P}, \mathcal{C}$ and gives $k_e, \mathcal{P}, \mathcal{C}$ to A. A may now submit an input string y to O, and O will return $D_{k_d}(y)$ to A. This is repeated as many time as A wants. Then A computes a plaintext $x \in \mathcal{P}$ and gives it to O. The oracle responds with $y_0 = E_{k_e}(x)$. A may now again submit an input string y to O, the only restriction is that y must be different from y_0. O will return $D_{k_d}(y)$ to A. This is repeated as many time as A wants.

Finally A outputs a bit b.

As before, we define $p_{A,i}(k)$ to be the probability that A outputs 1 in world i on input k. and the *advantage* of A to be

$$Adv_A(k) = |p_{A,0}(k) - p_{A,1}(k)|$$

DEFINITION 5 *We say that* (G, E, D) *is* chosen ciphertext (CCA)-secure*, if for all probabilistic polynomial time adversaries* A*, it holds that* $Adv_A(k)$ *is negligible in* k*.*

The general theorem from before can be expanded to this case as well:

THEOREM 5 *If there exists a family of trapdoor one-way permutations, then there exists a chosen ciphertext secure probabilistic public-key system.*

The construction behind this result leads to very inefficient systems. We would like to have more practical constructions, because chosen ciphertext security is not just a theoretical notion, but something we need to have in practice. Granted, it may seem that the attack we give the adversary in the CCA definition is very strong. One could ask whether this is too pessimistic - perhaps in real life, the adversary would not be in such a favorable situation?

However, there are in fact cases where the adversary has something that is close to a full CCA attack. For instance, many Internet servers can set up secure connections using the so called SSL protocol. Part of this is to decrypt a ciphertext received from a client. Until some years ago the system used was an RSA variant described in a (now outdated) version of the PKCS #1 standard. In this system, the decryption process could lead to different types of errors, if the input was not a legal ciphertext. If this occurred, servers would typically send an error message back to the client, specifying the type of error. But this means that an outsider now has access to (part of) the result of applying the decryption algorithm to an input he chooses. This is a a chosen ciphertext attack! Indeed, Bleichenbacher has shown that this can be used in practice to break the old version of PKCS #1.

Fortunately, we have today various heuristic methods one can use to get chosen ciphertext security efficiently. For example, in [2] the so called OAEP-system is proposed (Optimal Asymmetric Encryption Padding). OAEP is basically a general method for turning a public-key system with only deterministic security into a chosen ciphertext secure scheme. The basic idea is to encode a message m to be sent in special way before it is sent into the encryption algorithm. This means that only strings with a very special structure are ever encrypted. Put another way, if the original encryption algorithm can handle messages in some set $\mathcal{P}$, we will only use messages in some subset $\mathcal{P}'$. The decryption algorithm is defined such if after decryption, it finds that the result is not in $\mathcal{P}'$, it will simply output an error message. Ciphertexts that lead to plaintexts in $\mathcal{P}'$ are called *legal*.

Why would this help against a chosen ciphertext attack? well, if $\mathcal{P}'$ is very small compared to $\mathcal{P}$, then it may be a reasonable assumption that the only way in which the adversary could produce efficiently a legal ciphertext, is by choosing some plaintext m and encrypt it in the normal way. But if he submits this to the decryption oracle, he already knows the answer will be m. If he produces a ciphertext in any other way, by assumption it will be illegal with overwhelming probability. So again the adversary already knows what the oracle will say. Of course an oracle is not of any use if you can always predict in advance what it will answer! it might as well not be there and so we are back with the ciphertext only attack.

Concretely, OAEP works as follows: suppose the original encryption $E_{pk}(\cdot)$ works on k-bit strings. Then we choose two parameters k_0, k_1 such that $k_0 + k_1 < k$. The scheme can encrypt messages with n bits, where $n = k - k_0 - k_1$. We need two functions G, H, where $G : \{0,1\}^{k_0} \to \{0,1\}^{n+k_1}$

and $H : \{0,1\}^{n+k_1} \rightarrow \{0,1\}^{k_0}$. Usually, as G, H we use so called cryptographic hash functions, i.e., one-way functions that produce random looking output - they can be built, for instance, from symmetric encryption schemes and can therefore be very efficiently computable. We encrypt a message m as follows:

1. Choose $r \in \{0,1\}^{k_0}$ at random.
2. Compute $s = G(r) \oplus (m||0^{k_1})$, $t = H(s) \oplus r$, $w = s||t$, where $||$ means concatenation of strings.
3. Let the ciphertext be $E_{pk}(w)$.

For decryption, one use the secret key to reconstruct w and hence what should be s, t if the ciphertext was legal. Then we can find $r = t \oplus H(s)$, and finally $s \oplus G(r)$ should be some n-bit string m followed by k_1 0's. If this is not the case, the ciphertext was illegal, otherwise we output m.

No method of the OAEP type have been *proved* secure in the sense that their security follows only only from the RSA assumption, for instance. The problem is that it seems very difficult to design the encoding method such that one can prove that it is hard to generate legal ciphertexts wihtout knowing the plaintext. But they can nevertheless be quite adequate in practice, and OAEP is a part of many international standards for encryption, particularly in connection with RSA.

For systems based on discrete logarithms, state of the art is quite different. Cramer and Shoup [4] have built a quite practical system based on discrete logarithms which they can prove is chosen ciphertext secure under a reasonable assumption, the so called Decisional Diffie-Hellman asumption. This is the first practical system with such provable security. It is an open question if practical systems based on the RSA assumption with the same type of provable security exist.

References

[1] W.Alexi, B.Chor, O.Goldreich and C.P.Schnorr: *RSA and Rabin Functions: Certain parts are as hard as the Whole*, SIAM J.Computing, 17(1988), 194-209.

[2] Bellare and Rogaway: *Optimal Asymmetric Encryption*, Proc. of EuroCrypt 94, Springer Verlag LNCS series, 950.

[3] Bellare, Desai, Jokipii and Rogaway: *A concrete security treatment of symmetric encryption*, FOCS 97, full paper available from http://www-cse.ucsd.edu/users/mihir.

[4] Cramer and Shoup: *A Prcatical Public Key Cryptosystem Secure Against Adaptive Chosen Ciphertext Attacks*Proceedings of Crypto 98, Springer Verlag LNCS series 1462.