Provable-Security of Public-Key Encryption Schemes

Pooya Farshim

Contact: Room 2.07 or farshim@di.uminho.pt
Latest version available from: http://www.di.uminho.pt/∼farshim

Lecture 1: Factorisation and One-Way Functions

Intuition. Suppose that your lecturer writes the number 25927 on the board and asks everyone in the class for its prime factorisation. Immediately Alice, who is incidentally a friend of the lecturer, shouts 11×2357. What conclusion can be drawn from this? Can we conclude that Alice is very good at factoring integers? It might haven been the case that Alice's favourite primes are 11 and 2357 and hence she recognises 25927 as their product. To test Alice further we need to give her a few more tests. Furthermore, these tests should be in some sense random so they are unpredictable to Alice (hence she couldn't have prepared for them beforehand). Moreover, we need to check that alice does not take too long (i.e. she is "efficient") as we give her larger and larger numbers and that she does not make too many mistakes (i.e. her answer is incorrect "infrequently").

Efficient Algorithms. One way to formalise an "efficient" algorithm is thorough the notion of *probabilistic polynomial-time* (ppt) algorithms. A probabilistic algorithm is one which makes random choices during its computation. An algorithm $\mathcal{A}$ runs in polynomial time if its running time is bounded by a polynomial in the length of its input. Put differently, $\mathcal{A}(x)$ is poly-time if $\mathbf{Time}(\mathcal{A}(x)) \leq \text{poly}(|x|)$ where $|x|$ is the bit-length of x (here $\text{poly}(k)$ is a fixed polynomial (such as $10 \cdot k^2$). A ppt algorithm is therefore one which makes random choices and always terminates after a time which is at most $\text{poly}(|x|)$.

Frequency of success. To formalise the frequency of success of an algorithm, we look at the probability that it returns the correct answer. Here our sample space is over all possible inputs to the algorithm as well as random choices that $\mathcal{A}$ makes. Often in cryptography we want to say that frequency of success for some algorithm is "small". We define this notion through *negligible functions*. Informally, a function is negligible if it grows very slowly. More precisely, $f : \mathbb{N} \to \mathbb{R}$ is negligible if for any integer c there is an N, which may depend on c, such that for all $k > N$ we have that $|f(k)| < k^{-c}$. Therefore f is negligible if it goes to zero faster than inverse of of any polynomial.

Examples. 1) $f(k) = 2^{-k}$ is negligible; 2) $f(k) = 0$ is negligible; 3) $f(k) = -2^{-k}$ is negligible; 4) $f(k) = 1/k^{100}$ is not negligible; 4) $f(k) = 1/k^{\log k}$ is negligible; 5) Is $f(k) = k^{-k}$ negligible?

Formalising Hardness of Factorisation. Given the above two definitions (i.e. probabilistic polynomial-time algorithms and negligible functions) we can formalise what we mean by "factorisation is hard" as follows. A (hypothetical) challenger will choose two primes p and q, each having k bits, and sets $N = p \cdot q$. It then gives N to a potential factorisation algorithm $\mathcal{A}$. This algorithm can be probabilistic but should run in polynomial time. Algorithm $\mathcal{A}$ runs on N and finishes returning a guess (p', q') for the factors. We now need to check if this guess is correct i.e. if $p' \cdot q' = N$, $p' > 1$, and $q' > 1$. If so we declare that the algorithm $\mathcal{A}$ has *won*. Otherwise we declare that it has *lost*. The probability that $\mathcal{A}$ wins should be negligible as a function of k.

We refer to this type of interaction that we just described as an *experiment*. We write this experiment more precisely as a pseudo-code as shown in Figure 1. We say factorisation is hard, if the function (we will be defining the notation shortly):

$$\mathbf{Adv}^{\mathsf{fact}}_{\mathcal{A}}(k) := \Pr\left[\mathbf{Exp}^{\mathsf{FACT}}_{\mathcal{A}}(k) \Rightarrow \mathsf{true}\right]$$

is negligible as a function of k for any probabilistic polynomial-time algorithm $\mathcal{A}$.

Experiment $\mathbf{Exp}^{\mathsf{FACT}}_{\mathcal{A}}(k)$:
1. $p, q \xleftarrow{\$} \{p : p \text{ prime} \wedge |p| = k\}$
2. $N \leftarrow p \cdot q$
3. $(p', q') \xleftarrow{\$} \mathcal{A}(N)$
4. Return $(p' \cdot q' = N \wedge p' > 1 \wedge q' > 1)$

Fig. 1. Experiment defining the factorisation assumption.

The Experiment Notation. Let us explain some of the notation the we introduced in Figure 1. First look at the symbol $\mathbf{Exp}^{\mathsf{FACT}}_{\mathcal{A}}(k)$. Here FACT is the name of the experiment, we write this so we know which experiment we are talking about. We have $\mathcal{A}$ as a subscript which signifiers the fact that the experiment will use $\mathcal{A}$ in its code. Finally we have k, which is the length of the input we choose (this is also know as the security parameter: we expect things to get harder as we increase k). We write this as a function if k as we want to say the probability that outcome of the experiment is true (i.e. $\mathcal{A}$ has won) is negligible in k.

The symbol $\xleftarrow{\$}$ stands for the action of random sampling. Generally if S is a set, by $s \xleftarrow{\$} S$ we mean the action of sampling an element uniformly at random from S and assigning the result to s. Note that this implicitly means that there is an algorithm which performs this sampling action. Next we have the $\leftarrow$ symbol which, as usual, stands for assignment. The experiment then runs the algorithm $\mathcal{A}$, which might be probabilistic (hence we use $\xleftarrow{\$}$ rather than $\leftarrow$), and assigns its output to (p', q'). The experiment ends with a return statement which characterises when the algorithm $\mathcal{A}$ should be considered successful.

Finally, we have defined the advantage $\mathbf{Adv}^{\mathsf{fact}}_{\mathcal{A}}$ of $\mathcal{A}$ when run in this experiment. This is defined to be the probability that $\mathcal{A}$ is successful. Here by "$\Rightarrow \mathsf{true}$" we mean that the outcome of the experiment is true (i.e. $\mathcal{A}$ was successful). The probability is taken over all the randomness used inside the experiment (which include those of $\mathcal{A}$). Note that we have used a lower-case name in $\mathbf{Adv}^{\mathsf{fact}}_{\mathcal{A}}$. This is to distinguish that $\mathbf{Adv}$ is not an experiment, but rather a function related to factorisation.

One-way function. Informally a one-way function is a function which is easy to computer but hard to invert. One way to look at the FACT experiment is that it is hard to invert the function which takes two primes and multiplies them together. Let us generalise this to any function (rather than just multiplication). We say a function $f : \{0,1\}^* \to \{0,1\}^*$ is *one-way* if: 1) There exists a probabilistic polynomial-time algorithm F which on input x returns $f(x)$ (This is similar to the fact that multiplying two numbers can be performed efficiently. We also normally identify f with F, i.e. we just user f for F.); and 2) The advantage of any probabilistic polynomial-time algorithm

$\mathcal{A}$ in inverting f defined by

$$\mathbf{Adv}^{\mathsf{ow}}_{f,\mathcal{A}}(k) := \Pr\left[\mathbf{Exp}^{\mathsf{OW}}_{f,\mathcal{A}}(k) \Rightarrow \mathsf{true}\right]$$

is negligible as a function of the security parameter k. By Inverting f we mean the experiment shown in Figure 2.

Experiment $\mathbf{Exp}^{\mathsf{OW}}_{f,\mathcal{A}}(k)$:
1. $x \xleftarrow{\$} \{0,1\}^k$
2. $y \leftarrow f(x)$
3. $x' \xleftarrow{\$} \mathcal{A}(1^k, y)$
4. Return $(f(x') = y)$

Fig. 2. Experiment defining one-way property of a function f.

A Few Words on the Above Experiment. Note that again we are generating a random x and applying f to it. We give this to $\mathcal{A}$ and wait until it returns x'. Then we check if this is a correct answer, i.e. a correct pre-image. This is done by checking if $f(x') = f(x)$ in the last step. Note also the presence of the strange 1^k in line 3. 1^k in cryptography stands for $\mathtt{11}\cdots\mathtt{1}$ where there are k ones. Why is this needed? Look at the function which maps x to its bit-length. Is this function one-way according to the above definition? What if we drop 1^k from line 3? (Hint: $\mathcal{A}$ has to run in polynomial time in the length of its input).

Lecture 2: One-Way Public-Key Encryption Schemes

Public-Key Encryption Schemes

Background. Let us look at the RSA public-key encryption. Here the user secret key is (p, q, N) where $N = p \cdot q$ and its public key is (N, e) where e is such that $\gcd(e, (p-1)\cdot(q-1)) = 1$. To encrypt a message m, one computes the ciphertext $\mathsf{c} = \mathsf{m}^e \pmod N$. To decrypt, one first computes a d such that $ed = 1 \pmod{(p-1)(q-1)}$ and then returns $\mathsf{m} = \mathsf{c}^d \pmod N$. Note that this scheme is *correct* in the sense that if you generate a key pair properly and then encrypt a message m to get c, you will always recover m through decryption procedure if you pass c and the correct secret key to it. The algorithms which make up RSA constitute a specific set of encryption algorithms. In general, what we need are algorithms to generate keys, encrypt and decrypt. This is defined more formally next.

Syntax of a Public-Key Encryption Scheme. A public-key encryption scheme[1] $\Pi := (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a triple of (possibly probabilistic) polynomial-time algorithms as follows.

1. Algorithm $\mathsf{Gen}(1^k)$: This is the probabilistic key generation algorithm. It takes a security parameter 1^k and returns a tuple $(\mathsf{SK}, \mathsf{PK})$, where SK is the secret key and PK is the public key. We assume that PK contains 1^k so that later on when we pass PK to an algorithm we also implicitly pass 1^k.

[1] A *scheme* since a number of algorithms behave in a related way.

2. Algorithm $\mathsf{Enc}(\mathsf{m}, \mathsf{PK})$: This is the possibly probabilistic encryption algorithm. It takes as input a message m and a public key PK and returns a ciphertext c.
3. Algorithm $\mathsf{Dec}(\mathsf{c}, \mathsf{SK})$: This is the deterministic decryption algorithm. It takes a ciphertext c and a secret key SK and returns a message m or $\perp$. Here $\perp$ is a symbol which stands for "invalid ciphertext". It is special in the sense that it does not mean that the ciphertext decrypted to the message `invalid ciphertext`.

Correctness. We want an encryption scheme to be useful in the sense that decryption undoes encryption. More precisely, we require the advantage of the experiment in Figure 3 defined by

$$\mathbf{Adv}^{\mathsf{correct}}_{\Pi}(k) := \Pr\left[\mathbf{Exp}^{\mathsf{Correct}}_{\Pi}(k) \Rightarrow 1\right]$$

to be 1. Check that this experiment does say that "decryption undoes encryption"

Experiment $\mathbf{Exp}^{\mathsf{Correct}}_{\Pi}(k)$:
1. $(\mathsf{SK}, \mathsf{PK}) \xleftarrow{\$} \mathsf{Gen}(1^k)$
2. $\mathsf{m} \xleftarrow{\$} \mathsf{MsgSp}(\mathsf{PK})$ // Sampling from message space
3. $\mathsf{c} \xleftarrow{\$} \mathsf{Enc}(\mathsf{m}, \mathsf{PK})$
4. $\mathsf{m}' \leftarrow \mathsf{Dec}(\mathsf{c}, \mathsf{SK})$
5. Return $(\mathsf{m} = \mathsf{m}')$

Fig. 3. Experiment defining the correctness of an encryption scheme.

To allow a negligible failure in decryption (for instance because the keys are not correctly generated) we can let this advantage to be negligibly less than 1.

Message Space. For some encryption schemes the message space is simply $\{0,1\}^*$ or $\{0,1\}^\ell$ for some ℓ (which might depend on k but not the public key). However this is not true in general and the message space may depend on the public key. For instance in the RSA encryption scheme the message space is $\mathbb{Z}_N$. To deal which such schemes we also require the existence of the following algorithm.

- Algorithm $\mathsf{MsgSp}(\mathsf{m}, \mathsf{PK})$: This the deterministic message space membership algorithm. On input $\mathsf{m} \in \{0,1\}^*$ and a public key PK (returned by Gen) this algorithm returns either true or false. These indicate if the message passed to the algorithm is valid or invalid. We write $\mathsf{MsgSp}(\mathsf{PK})$ for the set of all $\mathsf{m} \in \{0,1\}^*$ which have $\mathsf{MsgSp}(\mathsf{m}, \mathsf{PK}) = \mathsf{true}$. As you see above we have also (ab)used $\mathsf{MsgSp}(\mathsf{PK})$ to do uniform sampling from $\mathsf{MsgSp}(\mathsf{PK})$[2].

Security of Public-Key Encryption Schemes

Let us think about what we can mean when we say "an encryption scheme is secure".

Intuition 1. One possible interpretation is that an encryption scheme is secure if "one cannot recover the secret key from the public key". However, this intuitive definition misses the point that encryption schemes are designed to provided confidentiality of messages and not that of the secret

[2] We assume that this space is finite so that the uniform distribution can be defined on it in a natural way.

key (although this should also be the case!). As a counterexample, take the encryption scheme which sets SK to be a long random string and sets the public key to be the empty sting. The encryption algorithm on input m returns the message itself and decryption on input a ciphertext returns the ciphertext itself. Check that this scheme is correct! It is easily to see that the advantage of any adversary (by an adversary we mean an algorithm which is attacking a cryptosystem) in guessing the secret key is 2^{-k}, a negligible function, but the scheme provides no security whatsoever.

Intuition 2. Another possible definition is that no efficient algorithm can recover the message encrypted under a ciphertext often. This definition does work to some extend (however as we shall see it has shortcomings). It is refereed to as one-wayness which we will be defining next.

Formal Definition of a One-Way Encryption Scheme. In line with the definition of a one-way function, we define the one-way property of a public-key encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ as follows. We say that scheme Π is one-way secure (under chosen-plaintext attacks; more on this coming later), written $\mathsf{OW\text{-}CPA}$, if the advantage of any probabilistic polynomial-time algorithm $\mathcal{A}$ defined by

$$\mathbf{Adv}^{\mathsf{ow\text{-}cpa}}_{\Pi,\mathcal{A}}(k) := \Pr\left[\mathbf{Exp}^{\mathsf{OW\text{-}CPA}}_{\Pi,\mathcal{A}}(k) \Rightarrow \mathsf{true}\right]$$

is negligible as a function k, where $\mathbf{Exp}^{\mathsf{OW\text{-}CPA}}_{\Pi,\mathcal{A}}(k)$ is the experiment shown in Figure 4.

Experiment $\mathbf{Exp}^{\mathsf{OW\text{-}CPA}}_{\Pi,\mathcal{A}}(k)$:
1. $(\mathsf{SK}, \mathsf{PK}) \xleftarrow{\$} \mathsf{Gen}(1^k)$
2. $\mathsf{m} \xleftarrow{\$} \mathsf{MsgSp}(\mathsf{PK})$
3. $\mathsf{c} \xleftarrow{\$} \mathsf{Enc}(\mathsf{m}, \mathsf{PK})$
4. $\mathsf{m}' \xleftarrow{\$} \mathcal{A}(\mathsf{PK}, \mathsf{c})$
5. Return $(\mathsf{m} = \mathsf{m}')$

Fig. 4. Experiment defining the one-wayness of a public-key encryption scheme.

Check that this experiment does capture the intuition that "given c it is hard to find m".

Lecture 3: Indistinguishability and Attack Models

Intuition 3. Recall the $\mathsf{OW\text{-}CPA}$ security definition from the last time. Suppose that an encryption leaks some information about the message but not necessarily all of it. For instance it can leak the first bit of the message. Would you classify this scheme as secure? Is it safe to do so? For instance, what happens if you encrypt your salary using this scheme? The adversary can check if it is above or below a certain threshold. What we ideally would like to have is that no information about the messaged should leaked from the ciphertext to any adversary. One way to formalise this is that we encrypt one of two known messages (of equal lengths[3]) and gives them to an adversary. The adversary should not be able to say which message was encrypted with probability better than guessing (i.e. 1/2). To see that this does imply the "no information leakage" property, if the encryption algorithm leaks information about messages then this leakage can be used to distinguish

[3] No encryption scheme can hide the length of the message. An encrypted 1 GB file is easily distinguishable from an encrypted 1 KB file, unless we pad the small file to 1GB in which case they become equal in length.

which messages is encrypted under a ciphertext. Conversely, if we can distinguish which messaged was encrypted, then in a sense the ciphertext is leaking some information (i.e. the information which is computed by the adversary).

Formal Definition 1. We say that a public-key encryption scheme has indistinguishable ciphertexts under chosen-plaintext attacks, written IND-CPA, if for any two messages message $\mathsf{m}_0, \mathsf{m}_1$ in the message space, the advantage of any probabilistic polynomial-time algorithm $\mathcal{A}$ defined by

$$\mathbf{Adv}^{\mathsf{ind\text{-}cpa}}_{\Pi,\mathcal{A},\mathsf{m}_0,\mathsf{m}_1}(k) := 2 \cdot \Pr\left[\mathbf{Exp}^{\mathsf{IND\text{-}CPA}}_{\Pi,\mathcal{A},\mathsf{m}_0,\mathsf{m}_1}(k) \Rightarrow \mathsf{true}\right] - 1$$

is negligible, where $\mathbf{Exp}^{\mathsf{IND\text{-}CPA}}_{\Pi,\mathcal{A},\mathsf{m}_0,\mathsf{m}_1}(k)$ is the experiment shown in Figure 5.

Experiment $\mathbf{Exp}^{\mathsf{OW\text{-}CPA}}_{\Pi,\mathcal{A},\mathsf{m}_0,\mathsf{m}_1}(k)$:
1. $(\mathsf{SK}, \mathsf{PK}) \xleftarrow{\$} \mathsf{Gen}(1^k)$
2. $b \leftarrow \{0,1\}$
3. $\mathsf{c} \xleftarrow{\$} \mathsf{Enc}(\mathsf{m}_b, \mathsf{PK})$
4. $b' \xleftarrow{\$} \mathcal{A}(\mathsf{PK}, \mathsf{c})$
5. Return $(b = b')$

Fig. 5. Experiment defining the indistinguishability of a public-key encryption scheme.

A Word About CPA. We name experiments in crypto using the template Goal-Model. By Goal we mean the goal that adversary is trying to achieve. For instance this can be trying to recover the whole plaintext (OW) or trying to distinguish which of two messages in encrypted under a ciphertext (IND). By Model we mean the attack model the adversary is running in. For instance, in the public-key scenario the adversary has access to public keys and can encrypt messages of his choice (note this is not true for symmetric encryption as the key is hidden). This is known as chosen-plaintext attack and is abbreviated to CPA. We will see another attack model, known as chosen-ciphertext attack, shortly.

Formal Definition of IND-CPA. To deal with message spaces with depend on public keys, we modify the above definition slightly, allowing the adversary to choose the two messages. We say that a public-key encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ has indistinguishable ciphertexts under chosen-plaintext attacks, written IND-CPA, if the advantage of any probabilistic polynomial-time algorithm $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ defined by

$$\mathbf{Adv}^{\mathsf{ind\text{-}cpa}}_{\Pi,\mathcal{A}}(k) := 2 \cdot \Pr\left[\mathbf{Exp}^{\mathsf{IND\text{-}CPA}}_{\Pi,\mathcal{A}}(k) \Rightarrow \mathsf{true}\right] - 1$$

is negligible, where $\mathbf{Exp}^{\mathsf{IND\text{-}CPA}}_{\Pi,\mathcal{A}}(k)$ is the experiment shown in Figure 6.

In the above st denotes some state information. What we are saying here is that we have an adversary $\mathcal{A}$ which runs in two stages. We start running the first stage of the adversary $\mathcal{A}_1$ on the public key. It pauses and outputs two messages. It also outputs some state information st which will be used to resume it. We then encrypt one of the messages at random and resume the adversary by giving it the state information st and the challenge ciphertext c.

Exercise. Show that no deterministic encryption scheme can be IND-CPA secure.

Experiment $\mathbf{Exp}^{\mathsf{IND\text{-}CPA}}_{\Pi,\mathcal{A}}(k)$:
1. $(\mathsf{SK}, \mathsf{PK}) \xleftarrow{\$} \mathsf{Gen}(1^k)$
2. $(\mathsf{m}_0, \mathsf{m}_1, \mathsf{st}) \xleftarrow{\$} \mathcal{A}_1(\mathsf{PK})$
3. $b \xleftarrow{\$} \{0,1\}$
4. $\mathsf{c} \xleftarrow{\$} \mathsf{Enc}(\mathsf{m}_b, \mathsf{PK})$
5. $b' \xleftarrow{\$} \mathcal{A}_2(\mathsf{c}, \mathsf{st})$
6. Return $(b = b')$

Fig. 6. Experiment defining the indistinguishability of a public-key encryption scheme.

Chosen-Ciphertext Attacks. In practice, the adversary might have attack capabilities other than just using the public key to encrypt various messages. For instance, it can submit ciphertexts of his choice to the user holding the secret key, and analyse the decryption behaviour of these ciphertexts. Let us give look at a more concrete example (taken from Katz's lecture notes).

Consider the following system for verifying credit cards. A user has a credit card number $B_1, B_2, B_3, \ldots, B_{48}$ (where each B_i represents one bit) which is encrypted, bit-wise, with the merchants public key PK and sent to the merchant as follows:

$$\mathsf{Enc}(B_1, \mathsf{PK}), \mathsf{Enc}(B_2, \mathsf{PK}), \mathsf{Enc}(B_3, \mathsf{PK}), \ldots, \mathsf{Enc}(B_{48}, \mathsf{PK}).$$

The merchant then immediately responds ACCEPT or REJECT, indicating whether the credit card is valid. Now, an adversary need not decrypt the message to recover the credit card: consider what happens if the first element of the above ciphertext is replaced by $\mathsf{Enc}(0, \mathsf{PK})$ (which an attacker can compute since the public key is available!) if the message is accepted by the merchant, the first bit of the credit card must be zero; if rejected, it is one. Continuing in this way, the adversary learns the entire credit card number after 48 such attempts.

Formal Definition. We say that a public-key encryption scheme has indistinguishable ciphertexts under chosen-ciphertext attacks, written $\mathsf{IND\text{-}CCA}$, if the advantage of any probabilistic polynomial-time algorithm $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ defined by

$$\mathbf{Adv}^{\mathsf{ind\text{-}cca}}_{\Pi,\mathcal{A}}(k) := 2 \cdot \Pr\left[\mathbf{Exp}^{\mathsf{IND\text{-}CCA}}_{\Pi,\mathcal{A}}(k) \Rightarrow \mathsf{true}\right] - 1$$

is negligible, where $\mathbf{Exp}^{\mathsf{IND\text{-}CCA}}_{\Pi,\mathcal{A}}(k)$ is the experiment shown in Figure 7.

Experiment $\mathbf{Exp}^{\mathsf{IND\text{-}CCA}}_{\Pi,\mathcal{A}}(k)$:
1. $(\mathsf{SK}, \mathsf{PK}) \xleftarrow{\$} \mathsf{Gen}(1^k)$
2. $(\mathsf{m}_0, \mathsf{m}_1, \mathsf{st}) \xleftarrow{\$} \mathcal{A}_1^{\mathbf{Decrypt}(\cdot)}(\mathsf{PK})$
3. $b \xleftarrow{\$} \{0,1\}$
4. $\mathsf{c} \xleftarrow{\$} \mathsf{Enc}(\mathsf{m}_b, \mathsf{PK})$
5. $b' \xleftarrow{\$} \mathcal{A}_2^{\mathbf{Decrypt}(\cdot)}(\mathsf{c}, \mathsf{st})$
6. Return $(b = b')$

Oracle $\mathbf{Decrypt}(\mathsf{c})$:
1. $\mathsf{m} \leftarrow \mathsf{Dec}(\mathsf{c}, \mathsf{SK})$
2. Return m

Fig. 7. Experiment defining the $\mathsf{IND\text{-}CCA}$ security of a public-key encryption scheme. The adversary $\mathcal{A}$ cannot query **Decrypt** on the ciphertext c given to it in the second stage (computed in line 4).

Remarks on IND-CCA. The experiment above is similar to the IND-CPA experiment except that the oracle **Decrypt** is introduced. An oracle can be thought of as a subroutine. It takes an input and returns on output. Furthermore, it is black-box in the sense that the adversary does not have access to its internal programme. The **Decrypt** oracle here takes a ciphertext c as input and returns the result of running $\mathsf{Dec}(\mathsf{c}, \mathsf{SK})$. The adversary has access to this oracle in both stages of its attack. However, it cannot query it in the stage with c. If this is allowed, then the adversary can trivially break any public-key encryption. What we are saying here is that adversary should not be able to distinguish which message is encrypted under a ciphertext even with the help of an oracle which decrypts any ciphertext of adversary's choice except that which he is supposed to "crack".

Remark on malleability. One implication of allowing access to the decryption oracle is that ciphertexts outputted by an IND-CCA secure encryption scheme are *non-malleable*. In other words, the adversary cannot take the ciphertext c given to him, somehow change it (malleate it) to a new ciphertext c' such that the message under c' is related to that under c. To see this, suppose the ciphertext is malleable. Then adversary can submit c' to the decryption oracle and get the result m'. Now if m' is related to m_0 the adversary will return 0 as its guess, otherwise it will return 1.

An Alternative Expression for the Advantage. The IND-CCA (and IND-CPA) advantage can be expressed in the following alternative way. Hre b denotes the bit chosen in the experiment in line 3 above.

$$
\begin{aligned}
\mathbf{Adv}^{\mathsf{ind\text{-}cca}}_{\Pi,\mathcal{A}}(k) &= 2\Pr\left[\mathbf{Exp}^{\mathsf{IND\text{-}CCA}}_{\Pi,\mathcal{A}}(k) \Rightarrow \mathsf{true}\right] - 1 \\
&= 2\Pr\left[\mathbf{Exp}^{\mathsf{IND\text{-}CCA}}_{\Pi,\mathcal{A}}(k) \Rightarrow \mathsf{true} \wedge b = 1\right] + 2\Pr\left[\mathbf{Exp}^{\mathsf{IND\text{-}CCA}}_{\Pi,\mathcal{A}}(k) \Rightarrow \mathsf{true} \wedge b = 0\right] - 1 \\
&= 2\Pr\left[\mathbf{Exp}^{\mathsf{IND\text{-}CCA}}_{\Pi,\mathcal{A}}(k) \Rightarrow \mathsf{true} | b = 1\right]\Pr\left[b = 1\right] + 2\Pr\left[\mathbf{Exp}^{\mathsf{IND\text{-}CCA}}_{\Pi,\mathcal{A}}(k) \Rightarrow \mathsf{true} | b = 0\right]\Pr\left[b = 0\right] - 1 \\
&= \Pr\left[\mathbf{Exp}^{\mathsf{IND\text{-}CCA}}_{\Pi,\mathcal{A}}(k) \Rightarrow \mathsf{true} | b = 1\right] + \Pr\left[\mathbf{Exp}^{\mathsf{IND\text{-}CCA}}_{\Pi,\mathcal{A}}(k) \Rightarrow \mathsf{true} | b = 0\right] - 1 \\
&= \Pr\left[\mathbf{Exp}^{\mathsf{IND\text{-}CCA}}_{\Pi,\mathcal{A}}(k) \Rightarrow \mathsf{true} | b = 1\right] - \left(1 - \Pr\left[\mathbf{Exp}^{\mathsf{IND\text{-}CCA}}_{\Pi,\mathcal{A}}(k) \Rightarrow \mathsf{true} | b = 0\right]\right) \\
&= \Pr\left[\mathbf{Exp}^{\mathsf{IND\text{-}CCA}}_{\Pi,\mathcal{A}}(k) \Rightarrow \mathsf{true} | b = 1\right] - \Pr\left[\mathbf{Exp}^{\mathsf{IND\text{-}CCA}}_{\Pi,\mathcal{A}}(k) \Rightarrow \mathsf{false} | b = 0\right] \\
&= \Pr\left[\mathcal{A} \text{ returns } b' = 1 | b = 1\right] - \Pr\left[\mathcal{A} \text{ returns } b' = 1 | b = 0\right]
\end{aligned}
$$

Lets see what this is saying: we are measuring how the output behaviour of $\mathcal{A}$ changes when the bit b is flipped from 1 to 0. The separation of different events happening in an experiment ($b = 0$ or $b = 1$ here) is often useful when proving security of a cryptosystem.

Lecture 4: Discrete Logarithms and the ElGamal Encryption Scheme

Background on Groups. Groups are sets where one can multiply two elements together in a natural way. By multiply here we mean any general way of combining two elements together (e.g. adding them!). The set also has an "identity element" which when multiplied with any element leaves it unchanged (0 for addition). For any element in the set one can find its "inverse" which when multiplied by it gives the identity element (this is the negation of a number for addition: $2+(-2) = 0$). Furthermore the order of applying the rule to elements does not matter ($2+3 = 3+2$ and $2+(3+4) = (2+3)+4$). The order of a group is the number of elements in the underlying set. A

generator of a group is an element which if repeatedly multiplied with itself gives the whole group. If you are not satisfied with these informal reminders, look up the formal definition on Wikipedia!

Group Schemes. A group scheme is a set of given algorithms to perform various computations with the group elements. These algorithms are for: 1) Multiplying two elements; 2) Dividing two elements (multiplying one by the inverse of the other); 3) Checking if a string encodes a group element; 4) Computing a generator; and 5) Sampling random elements from the group. We do not need to know more than this about group schemes. Just think of them as "groups which one can concretely work with on a computer". We will denote these by $(\mathbb{G}, g, p)$ with $\mathbb{G}$ denoting the descriptions of various algorithms, g the generator, and p the order of the group (which will be prime). We will need to generate groups of different sizes according to the value of the security parameter k. We denote by $\mathcal{GP}$ an algorithm which takes 1^k and returns $(\mathbb{G}, g, p)$ with p a k-bit prime. In a group of prime order p every non-identity element is a generator.

The Discrete Logarithm Assumption. Informally this assumption says that in a group $\mathbb{G}$ with generator g, given g^x, it is hard to find x. More precisely, we say a group scheme $\mathcal{GP}$ satisfies the discrete-logarithm assumption if the advantage of any probabilistic polynomial-time algorithm $\mathcal{A}$ defined by

$$\mathbf{Adv}^{\mathsf{dl}}_{\mathcal{GP},\mathcal{A}}(k) := \Pr\left[\mathbf{Exp}^{\mathsf{DL}}_{\mathcal{GP},\mathcal{A}}(k) \Rightarrow \mathsf{true}\right]$$

is negligible as function of k, where $\mathbf{Exp}^{\mathsf{DL}}_{\mathcal{GP},\mathcal{A}}(k)$ is the experiment shown in Figure 8.

Experiment $\mathbf{Exp}^{\mathsf{DL}}_{\mathcal{GP},\mathcal{A}}(k)$:
1. $(\mathbb{G}, g, p) \xleftarrow{\$} \mathcal{GP}(1^k)$
2. $x \xleftarrow{\$} \mathbb{Z}_p$; $h \leftarrow g^x$
3. $x' \xleftarrow{\$} \mathcal{A}((\mathbb{G}, g, p), h)$
4. Return $(x = x')$

Fig. 8. Experiment defining the discrete logarithm assumption.

Non-Example. Take the group of integers modulo p a prime under addition with generator 1. What is the discrete logarithm problem? Is it hard?

The Computational Diffie-Hellman Assumption. Informally this assumption says that in a group $\mathbb{G}$ with generator g, given (g^x, g^y), it is hard to find g^{xy}. More precisely, we say a group scheme $\mathcal{GP}$ satisfies the discrete-logarithm assumption if the advantage of any probabilistic polynomial-time algorithm $\mathcal{A}$ defined by

$$\mathbf{Adv}^{\mathsf{cdh}}_{\mathcal{GP},\mathcal{A}}(k) := \Pr\left[\mathbf{Exp}^{\mathsf{CDH}}_{\mathcal{GP},\mathcal{A}}(k) \Rightarrow \mathsf{true}\right]$$

is negligible as function of k, where $\mathbf{Exp}^{\mathsf{CDH}}_{\mathcal{GP},\mathcal{A}}(k)$ is the experiment shown in Figure 9.

The Decisional Diffie-Hellman Assumption. Informally this assumption says that in a group $\mathbb{G}$ with generator g, given a tuple of the form (g^x, g^y, g^z) or one of the form (g^x, g^y, g^{xy}), it is hard to decide which was given. More precisely, we say a group scheme $\mathcal{GP}$ satisfies the decisional Diffie-Hellman assumption if the advantage of any probabilistic polynomial-time algorithm $\mathcal{A}$ defined

Experiment $\mathbf{Exp}^{\mathsf{CDH}}_{\mathcal{GP},\mathcal{A}}(k)$:
1. $(\mathbb{G}, g, p) \xleftarrow{\$} \mathcal{GP}(1^k)$
2. $x \xleftarrow{\$} \mathbb{Z}_p$; $y \xleftarrow{\$} \mathbb{Z}_p$
3. $g_1 \leftarrow g^x$; $g_2 \leftarrow g^y$
4. $h \xleftarrow{\$} \mathcal{A}((\mathbb{G}, g, p), g_1, g_2)$
5. Return $(h = g^{xy})$

Fig. 9. Experiment defining the computational Diffie-Hellman assumption.

by

$$\mathbf{Adv}^{\mathsf{ddh}}_{\mathcal{GP},\mathcal{A}}(k) := 2 \cdot \Pr\left[\mathbf{Exp}^{\mathsf{DDH}}_{\mathcal{GP},\mathcal{A}}(k) \Rightarrow \mathsf{true}\right] - 1$$

is negligible as function of k, where $\mathbf{Exp}^{\mathsf{DDH}}_{\mathcal{GP},\mathcal{A}}(k)$ is the experiment shown in Figure 10.

Experiment $\mathbf{Exp}^{\mathsf{DDH}}_{\mathcal{GP},\mathcal{A}}(k)$:
1. $(\mathbb{G}, g, p) \xleftarrow{\$} \mathcal{GP}(1^k)$
2. $x \xleftarrow{\$} \mathbb{Z}_p$; $y \xleftarrow{\$} \mathbb{Z}_p$; $z \xleftarrow{\$} \mathbb{Z}_p$
3. $g_1 \leftarrow g^x$; $g_2 \leftarrow g^y$
4. $b \xleftarrow{\$} \{0, 1\}$
5. If $b = 0$ Then $g_3 \leftarrow g^z$ Else $g_3 \leftarrow g^{xy}$
6. $b' \xleftarrow{\$} \mathcal{A}((\mathbb{G}, g, p), g_1, g_2, g_3)$
7. Return $(b = b')$

Fig. 10. Experiment defining the decisional Diffie-Hellman assumption.

The Relation Between DL and CDH. Note that if the CDH assumption holds with respect to $\mathcal{GP}$, then so does the DL assumption. To see this, note that if there exists an algorithm $\mathcal{A}$ which solves the DL problem, then there is an algorithm $\mathcal{B}$ which solves the CDH problem. How? Algorithm $\mathcal{B}$ on input $((\mathbb{G}, g, p), g_1, g_2)$ first computes the discrete logarithm of g_1 using $\mathcal{A}$ (i.e. runs $\mathcal{A}$ on $((\mathbb{G}, g, p), g_1)$ and gets a value x). Then it raises g_2 to power x to get h and outputs it. If x is the correct value of the discrete logarithm, then h will also be the correct answer to the CDH problem. We have therefore shown that for any algorithm $\mathcal{A}$ solving DL there is an algorithm $\mathcal{B}$ solving CDH such that:

$$\mathbf{Adv}^{\mathsf{cdh}}_{\mathcal{GP},\mathcal{B}}(k) = \mathbf{Adv}^{\mathsf{dl}}_{\mathcal{GP},\mathcal{A}}(k)$$

Put differently, if the CDH holds, then the left hand side of the above equality is negligible and therefore so is the right hand side. Since $\mathcal{A}$ was arbitrary we conclude that the DL assumption also holds (advantage of any $\mathcal{A}$ is negligible in solving DL).

The Relation Between CDH and DDH. Now we show that if the DDH assumption holds with respect to $\mathcal{GP}$, then so does the CDH assumption. To see this, note that if there exists an algorithm $\mathcal{A}$ which solves the DDH problem, then there is an algorithm $\mathcal{B}$ which solves the CDH problem as follows. Algorithm $\mathcal{B}$ on input $((\mathbb{G}, g, p), g_1, g_2, g_3)$ first computes the "CDH of g_1 and g_2" by running $\mathcal{A}$ on $((\mathbb{G}, g, p), g_1, g_2)$ and gets a value h. Then it checks if $h = g_3$. If this holds it returns 1, and otherwise it returns 0.

If h is computed correctly by $\mathcal{A}$ then $\mathcal{B}$'s answer will also be correct expect if the bit chosen in the DDH game was 0 and the random element g^z happened to be g^{xy}. The probability of this event will be small. More formally,

$$\begin{aligned}
\mathbf{Adv}^{\mathsf{ddh}}_{\mathcal{GP},\mathcal{B}}(k) &= 2 \cdot \Pr\left[\mathbf{Exp}^{\mathsf{DDH}}_{\mathcal{GP},\mathcal{B}}(k) \Rightarrow \mathsf{true}\right] - 1 \\
&= 2 \cdot \Pr\left[\mathbf{Exp}^{\mathsf{DDH}}_{\mathcal{GP},\mathcal{B}}(k) \Rightarrow \mathsf{true} | b = 1\right] \cdot (1/2) + 2 \cdot \Pr\left[\mathbf{Exp}^{\mathsf{DDH}}_{\mathcal{GP},\mathcal{B}}(k) \Rightarrow \mathsf{true} | b = 0\right] \cdot (1/2) - 1 \\
&= \Pr\left[\mathbf{Exp}^{\mathsf{DDH}}_{\mathcal{GP},\mathcal{B}}(k) \Rightarrow \mathsf{true} | b = 1\right] - \Pr\left[\mathbf{Exp}^{\mathsf{DDH}}_{\mathcal{GP},\mathcal{B}}(k) \Rightarrow \mathsf{false} | b = 0\right] \\
&= \Pr\left[\mathcal{B} \text{ returns } 1 | b = 1\right] - \Pr\left[\mathcal{B} \text{ returns } 1 | b = 0\right] \\
&= \Pr\left[\mathcal{A} \text{ returns } \mathsf{CDH}\right] - \Pr\left[\mathcal{A} \text{ returns } h | \, g_3 \xleftarrow{\$} \mathbb{G}\right] \\
&= \Pr\left[\mathbf{Exp}^{\mathsf{CDH}}_{\mathcal{GP},\mathcal{A}}(k) \Rightarrow \mathsf{true}\right] - 1/p \\
&= \mathbf{Adv}^{\mathsf{cdh}}_{\mathcal{GP},\mathcal{A}}(k) - 1/p.
\end{aligned}$$

The fifth equality follows from the fact that for any (even unbounded) $\mathcal{A}$, the probability of guessing a completely random g_3 which is independent of $\mathcal{A}$'s view is $1/p$. Rearranging, we have shown that for any algorithm $\mathcal{A}$ solving CDH there is an algorithm $\mathcal{B}$ solving DDH such that:

$$\mathbf{Adv}^{\mathsf{ddh}}_{\mathcal{GP},\mathcal{B}}(k) + \frac{1}{p} = \mathbf{Adv}^{\mathsf{cdh}}_{\mathcal{GP},\mathcal{A}}(k)$$

Put differently, if the DDH holds and p is large, then the left hand side of the above equality is negligible and therefore so is the right hand side. Since $\mathcal{A}$ was arbitrary we conclude that the CDH assumption also holds (advantage of any $\mathcal{A}$ is negligible in solving CDH).

This is the general approach when proving theorems in cryptography. We take an arbitrary ppt algorithm attacking a cryptosystem. We then use $\mathcal{A}$ as a subroutine to solve another problem which we have assumed to be hard. This is a contraction, and hence the algorithm $\mathcal{A}$ could not have existed originally, i.e. the cryptosystem is secure.

The ElGamal Encryption Scheme. Given a computational group scheme $\mathcal{GP}$, the ElGamal public-key encryption is as shown in Figure 11.

Algorithm $\mathsf{Gen}(1^k)$:
1. $(\mathbb{G}, g, p) \xleftarrow{\$} \mathcal{GP}(1^k)$
2. $x \xleftarrow{\$} \mathbb{Z}_p; h \leftarrow g^x$
3. $\mathsf{SK} \leftarrow ((\mathbb{G}, g, p), x)$
4. $\mathsf{PK} \leftarrow ((\mathbb{G}, g, p), h)$
5. Return $(\mathsf{SK}, \mathsf{PK})$

Algorithm $\mathsf{Enc}(\mathsf{m}, \mathsf{PK})$:
1. Parse $(\mathbb{G}, g, p, h) \leftarrow \mathsf{PK}$
2. $r \xleftarrow{\$} \mathbb{Z}_p$; $\mathsf{c}_1 \leftarrow g^r$
3. $\mathsf{c}_2 \leftarrow \mathsf{m} \cdot h^r$
4. $\mathsf{c} \leftarrow (\mathsf{c}_1, \mathsf{c}_2)$
5. Return c

Algorithm $\mathsf{Dec}(\mathsf{c}, \mathsf{SK})$:
1. Parse $(\mathbb{G}, g, p, x) \leftarrow \mathsf{SK}$
2. Parse $(\mathsf{c}_1, \mathsf{c}_2) \leftarrow \mathsf{c}$
3. $\mathsf{m} \leftarrow \mathsf{c}_2 / (\mathsf{c}_1)^x$
4. Return m

Fig. 11. The ElGamal public-key encryption scheme.

Exercise. Check that this scheme is correct.

Lecture 5: Security of the ElGamal Encryption Scheme

One-wayness under the CDH Assumption. We now prove that if the CDH assumption holds with respect to $\mathcal{GP}$, then the ElGamal encryption scheme is one-way. This will be our first security proof. More precisely:

Theorem 1. *Let $\mathcal{A}$ be a probabilistic polynomial-time algorithm against the ElGamal encryption scheme in the* OW-CPA *sense. Then there is a probabilistic polynomial-time algorithm $\mathcal{B}$ against $\mathcal{GP}$ solving the* CDH *problem such that:*

$$\mathbf{Adv}^{\mathsf{cdh}}_{\mathcal{GP},\mathcal{B}}(k) = \mathbf{Adv}^{\mathsf{ow\text{-}cpa}}_{\Pi,\mathcal{A}}(k).$$

In particular, if the CDH *assumption holds with respect to $\mathcal{GP}$, then the left hand side of the above equality is a negligible function, and consequently so is the right hand side. This means ElGamal is one-way as $\mathcal{A}$ was an arbitrary probabilistic polynomial-time algorithm.*

Proof. The algorithm $\mathcal{B}$ operates as shown in Figure 12 below.

Algorithm $\mathcal{B}((\mathbb{G}, g, p), g_1, g_2)$:
1. $\mathsf{PK} \leftarrow ((\mathbb{G}, g, p), g_1)$ // here implicitly $g_1 = g^x$
2. $\mathsf{c}_1 \leftarrow g_2$ // here implicitly $g_2 = g^y$
3. $t \xleftarrow{\$} \mathbb{Z}_p$; $\mathsf{c}_2 \leftarrow g^t$ // here implicitly $\mathsf{m} = \mathsf{c}_2/g^{xy}$
4. $\mathsf{c} \leftarrow (\mathsf{c}_1, \mathsf{c}_2)$
5. Run $\mathcal{A}(\mathsf{PK}, \mathsf{c})$
6. $\mathcal{A}$ finishes returning a message m'
7. $h \leftarrow \mathsf{c}_2/\mathsf{m}'$
8. Return h

Fig. 12. CDH adversary $\mathcal{B}$ against $\mathcal{GP}$ based on an OW-CPA adversary $\mathcal{A}$ against ElGamal.

Note that if $\mathcal{A}$ is ppt, then so is $\mathcal{B}$. Let us analyse the success probability of $\mathcal{B}$ based on the success probability of $\mathcal{A}$.

$$\begin{aligned}
\mathbf{Adv}^{\mathsf{cdh}}_{\mathcal{GP},\mathcal{B}}(k) &= \Pr\left[\mathbf{Exp}^{\mathsf{CDH}}_{\mathcal{GP},\mathcal{B}}(k) \Rightarrow \mathsf{true}\right] && (1)\\
&= \Pr\left[h = g^{xy}\right] && (2)\\
&= \Pr\left[\mathsf{c}_2/\mathsf{m}' = g^{xy}\right] = \Pr\left[\mathsf{m}' = \mathsf{c}_2/g^{xy}\right] = \Pr\left[\mathsf{m}' = \mathsf{m}\right] && (3)\\
&= \Pr\left[\mathbf{Exp}^{\mathsf{OW\text{-}CPA}}_{\Pi,\mathcal{A}}(k) \Rightarrow \mathsf{true}\right] && (4)\\
&= \mathbf{Adv}^{\mathsf{ow\text{-}cpa}}_{\Pi,\mathcal{A}}(k). && (5)
\end{aligned}$$

Let us explain why each of the above equalities hold.

- Equality (1): By definition of advantage.
- Equality (2): By definition of algorithm $\mathcal{B}$.
- Equalities in (3): By implicit values (i.e. $\mathcal{B}$ does not need access to the explicit values) of x, y and m (See Figure 12). Note that we know there are values x and y such that $g_1 = g^x$, $g_2 = g^y$. We have also implicitly defined m to be g^t/g^{xy} for a random t.

– Equality (6): This is the conceptually important step. We are saying here that the probability that $\mathsf{m}' = \mathsf{m}$ when run on the public key and ciphertext as constructed by $\mathcal{B}$ above is same as running $\mathcal{A}$ in the OW-CPA experiment. Let us check this.
  - In the OW-CPA experiment $(\mathbb{G}, g, p)$ is a generated by $\mathcal{GP}(1^k)$. So is the value passed by $\mathcal{B}$ to $\mathcal{A}$ above as $(\mathbb{G}, g, p)$ is generated by $\mathcal{GP}(1^k)$ in the CDH experiment.
  - In the OW-CPA experiment the public key is a random element of $\mathbb{G}$ (together with $(\mathbb{G}, g, p)$). So is the value passed by $\mathcal{B}$ to $\mathcal{A}$ above as the public key since g_1 is generated randomly in the CDH experiment.
  - In the OW-CPA experiment the c_1 is a random element of $\mathbb{G}$. So is the value passed by $\mathcal{B}$ to $\mathcal{A}$ above as c_1 since g_2 is generated randomly in the CDH experiment.
  - In the OW-CPA experiment the c_2 is a random element of $\mathbb{G}$ (a message m) multiplied by PK^r. This is itself a uniform element on $\mathbb{G}$ generated. So is the value passed by $\mathcal{B}$ to $\mathcal{A}$ above as c_2 since g^t is generated randomly. Note that the implicit message here is $\mathsf{m} = g^t/g^{xy}$.
– Equality (7): By definition of advantage.

□

Indistinguishability under the DDH Assumption. We now prove that if the DDH assumption holds with respect to $\mathcal{GP}$, then the ElGamal encryption scheme is indistinguishable under chosen-plaintext attacks, i.e. it is IND-CPA secure.

Theorem 2. *Let $\mathcal{A}$ be a probabilistic polynomial-time against the ElGamal encryption scheme in the* IND-CPA *sense. Then there is a probabilistic polynomial-time algorithm $\mathcal{B}$ against $\mathcal{GP}$ solving the* DDH *problem such that:*

$$\mathbf{Adv}^{\mathsf{ddh}}_{\mathcal{GP},\mathcal{B}}(k) = 1/2 \cdot \mathbf{Adv}^{\mathsf{ind\text{-}cpa}}_{\Pi,\mathcal{A}}(k).$$

In particular, if the DDH *assumption holds with respect to $\mathcal{GP}$, then the left hand side of the above equality is a negligible function, and consequently so is the right hand side. This means ElGamal ciphertexts are indistinguishable since $\mathcal{A}$ was an arbitrary probabilistic polynomial-time algorithm.*

The intuition behind the proof is this: suppose (g^x, g^y, h) is given where either $h = g^z$ or $h = g^{xy}$. The idea is that depending on h we can generate a ciphertext which bears no information about the underlying message (when $h = g^z$) or is a correctly formed ciphertext (when $h = g^{xy}$). No adversary can guess which message is encrypted in the first case, but a successful IND-CPA adversary $\mathcal{A}$ will have a noticeable advantage in the second case. We can thus use this difference in behaviour of $\mathcal{A}$ to detect if $h = g^z$ or $h = g^{xy}$.

Proof. The algorithm $\mathcal{B}$ operates as shown in Figure 12.

Note that if $\mathcal{A}$ is ppt, then so is $\mathcal{B}$. Let us analyse the success probability of $\mathcal{B}$ based on the success probability of $\mathcal{A}$. In the following b denotes the bit chosen in the DDH experiment.

$$\begin{aligned}
\mathbf{Adv}^{\mathsf{ddh}}_{\mathcal{GP},\mathcal{B}}(k) &= 2 \cdot \Pr\left[\mathbf{Exp}^{\mathsf{DDH}}_{\mathcal{GP},\mathcal{B}}(k) \Rightarrow \mathsf{true}\right] - 1 && (6)\\
&= \Pr\left[\mathcal{A} \text{ returns } d' = d | b = 1\right] + \Pr\left[\mathcal{A} \text{ returns } d' = d | b = 0\right] - 1 && (7)\\
&= \Pr\left[\mathbf{Exp}^{\mathsf{IND\text{-}CPA}}_{\Pi,\mathcal{A}}(k) \Rightarrow \mathsf{true}\right] + 1/2 - 1 && (8)\\
&= 1/2 \cdot \mathbf{Adv}^{\mathsf{ind\text{-}cpa}}_{\Pi,\mathcal{A}}(k). && (9)
\end{aligned}$$

Let us explain why each of the above equalities hold.

Algorithm $\mathcal{B}((\mathbb{G}, g, p), g_1, g_2, g_3)$:
1. $\mathsf{PK} \leftarrow ((\mathbb{G}, g, p), g_1)$
2. Run $\mathcal{A}_1(\mathsf{PK})$
3. $\mathcal{A}_1$ finishes returning $(\mathsf{m}_0, \mathsf{m}_1, \mathsf{st})$
4. $d \xleftarrow{\$} \{0, 1\}$
5. $\mathsf{c}_1 \leftarrow g_2$; $\mathsf{c}_3 \leftarrow \mathsf{m}_d \cdot g_3$
6. $\mathsf{c} \leftarrow (\mathsf{c}_1, \mathsf{c}_2)$
7. Run $\mathcal{A}_2(\mathsf{c}, \mathsf{st})$
8. $\mathcal{A}_2$ finishes returning a bit d'
9. Return $(d = d')$

Fig. 13. Algorithm $\mathcal{B}$ solving DDH problem based on $\mathcal{A}$ attacking ElGamal indistinguishability.

- Equality (6): By definition of advantage.
- Equality (7): Using the fact that experiment returns true if and only if $d = d'$ and conditioning on the value of b. This was derived at the end of Lecture 3.
- Equality (8): This is the main step of the proof. There are two equalities here to be shown, namely:

$$\Pr\left[\mathcal{A} \text{ returns } d' = d | b = 1\right] = \Pr\left[\mathbf{Exp}^{\mathsf{IND\text{-}CPA}}_{\Pi,\mathcal{A}}(k) \Rightarrow \mathsf{true}\right] \text{ and}$$
$$\Pr\left[\mathcal{A} \text{ returns } d' = d | b = 0\right] = 1/2.$$

  Lets first look at the second of these. Note that the public key passed by $\mathcal{B}$ to $\mathcal{A}$ is distributed correctly. Suppose the bit chosen in the DDH experiment is $b = 0$. Then g_3 is a random element of the $\mathbb{G}$. In this case *the view of $\mathcal{A}$ is independent of the bit d chosen by $\mathcal{B}$* as the ciphertext given to $\mathcal{A}$ is simply a randomly distributed element in $\mathbb{G} \times \mathbb{G}$ and hence bears no information on d. The probability that any (even unbounded) $\mathcal{A}$ outputs d' with $d' = d$ is therefore $1/2$[4].
  Let us now look at the first equality. Note that the public key passed by $\mathcal{B}$ to $\mathcal{A}$ is distributed correctly. Suppose the bit chosen in the DDH experiment is $b = 1$. Then $g_3 = g^{xy}$, where x and y are such that $g_1 = g^x$ and $g_2 = g^y$. Now be examining the code of $\mathcal{B}$ and that of the DDH experiment (when $b = 1$) we see that $\mathcal{A}$ is run in exactly the same way as it would be run in the IND-CPA experiment. More precisely, when $g_3 = g^{xy}$ then $\mathsf{c}_2 = \mathsf{m}_d \cdot g_3^{xy}$, $\mathsf{c}_1 = g^y$, and $\mathsf{PK} = (\mathbb{G}, g, p, g^x)$. Bit d maps to the bit chosen in the IND-CPA experiment. Hence when $\mathcal{B}$ runs $\mathcal{A}$, algorithm $\mathcal{A}$ first gets a correctly distributed public key. It then outputs two messages, and gets a correctly formed and distributed ciphertext encrypting one at random according to d. Furthermore, $d = d'$ is the same condition as that which makes the IND-CPA experiment to have outcome true.
- Equality (9): By definition of advantage.

□

A Chosen-Ciphertext Attack on ElGamal. Try and show that ElGamal is insecure under chosen-ciphertext attacks. To this end, construct an algorithm $\mathcal{A}$ which on input a ciphertext c and a public key PK, modifies it to get a new ciphertext $\mathsf{c}' \neq \mathsf{c}$ such that the decryption oracle when

[4] Note that although the ciphertext passed to $\mathcal{A}$ is not valid, algorithm $\mathcal{A}$ still has to run on it an return a bit. Its behaviour will not be affected by d as it is completely hidden from its view

queried on c' helps him to distinguish which message was encrypted under c (i.e. malleate c to c'). This algorithm is shown in Figure 14. Show that as long as the message space has at least two elements, $\mathcal{A}$ is successful with probability 1. In particular you need to show that in step 4 a valid ciphertext is constructed.

Algorithm $\mathcal{A}_1(\mathsf{PK})$:
1. $\mathsf{m}_0 \leftarrow \mathsf{MsgSp}(\mathsf{PK})$
2. $\mathsf{m}_1 \leftarrow \mathsf{MsgSp}(\mathsf{PK})$ such that $\mathsf{m}_1 \neq \mathsf{m}_0$
3. $\mathsf{st} \leftarrow \mathsf{PK}$
4. Return $(\mathsf{m}_0, \mathsf{m}_1, \mathsf{st})$

Algorithm $\mathcal{A}_2(\mathsf{c}, \mathsf{st})$:
1. Parse $\mathsf{PK} \leftarrow \mathsf{st}$
2. Parse $(\mathsf{c}_1, \mathsf{c}_2) \leftarrow \mathsf{c}$
3. $r \xleftarrow{\$} \mathbb{Z}_p - \{0\}$
4. $\mathsf{c}'_1 \leftarrow \mathsf{c}_1 \cdot g^r$; $\mathsf{c}'_2 \leftarrow \mathsf{c}_2 \cdot (\mathsf{PK})^r$; $\mathsf{c}' \leftarrow (\mathsf{c}'_1, \mathsf{c}'_2)$
5. Query **Decrypt**(c). Receive m.
6. If $\mathsf{m} = \mathsf{m}_0$ Return 1, Else Return 0

Fig. 14. Algorithm $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ lunching an IND-CCA attack against ElGamal.

Construction of IND-CCA schemes. These are quite a bit harder to construct. The first practical and IND-CCA-secure encryption scheme was given by Cramer and Shoup. The authors show that their scheme is IND-CCA-secure if the DDH assumption holds.

Bibliography

Jonathan Katz's lecture notes on introductory cryptography are a good (and free) place to read more on provable security.

`http://www.cs.umd.edu/~jkatz/TEACHING/crypto_F02/lectures.html`

His book with Yehuda Lindell presents his lecture notes in a more systematic and comprehensive way. There are copies of this book available at the university library. Chapter 10 is on public-key encryption.

`http://www.cs.umd.edu/~jkatz/imc.html`

You can find many other lecture notes on crypto online.