Chapter 20

The Diffie-Hellman Problem and Cryptographic Applications

This chapter introduces some basic applications of the discrete logarithm problem in cryptography, such as Diffie-Hellman key exchange and "textbook" Elgamal encryption. A brief security analysis of these systems is given. This motivates the computational and decisional Diffie-Hellman problems (CDH and DDH). A thorough discussion of these computational problems will be given in Chapter 21.

20.1 The Discrete Logarithm Assumption

The discrete logarithm problem (DLP) was defined in Definition 13.0.2. Our main interest is the DLP in an algebraic group or algebraic group quotient over a finite field $\mathbb{F}_q$ (for example, elliptic curves, the multiplicative group of a finite fields, tori etc). We always use multiplicative notation for groups in this chapter. As discussed in Section 13.2, in practice we usually restrict to groups of prime order r.

Recall that the difficulty of the DLP is defined with respect to an instance generator that runs on input a security parameter κ. An algorithm to solve the DLP with respect to a given instance generator is only required to succeed with a noticeable probability. The **discrete logarithm assumption** is that there exist instance generators that, on input κ, output instances of the DLP such that no algorithm A running in polynomial-time in κ can solve the DLP apart from with negligible (in κ) probability. The cryptosystems in this chapter rely on the discrete logarithm assumption (and other assumptions).

20.2 Key Exchange

20.2.1 Diffie-Hellman Key Exchange

The starting point of discrete logarithms (indeed, of public key cryptography) is the seminal paper of Diffie and Hellman [178] from 1976 (more recently it became known that this idea was also found by Williamson at GCHQ in 1974).

Suppose Alice and Bob want to agree on a random key K. Assume they both know an algebraic group or algebraic group quotient G and some element $g \in G$ of prime order r (everyone in the world could use the same g). They perform the following protocol:

- Alice chooses a random integer $0 < a < r$ and sends $\mathsf{c}_1 = g^a$ to Bob.
- Bob chooses a random integer $0 < b < r$ and sends $\mathsf{c}_2 = g^b$ to Alice.
- On receiving c_2 Alice computes $K = \mathsf{c}_2^a$.
- On receiving c_1 Bob computes $K = \mathsf{c}_1^b$.

Hence, both players share the key $K = g^{ab}$. One can derive (see Definition 20.2.10 below) a bitstring from the group element K for use as the key of a symmetric encryption scheme. Hence, encryption of data or other functionalities can be implemented using traditional symmetric cryptography. The key K is called the **session key** and the values $\mathsf{c}_1, \mathsf{c}_2$ in the protocol are called **messages** or **ephemeral keys**.

We discuss the security of key exchange protocols (in particular, person-in-the-middle attacks and authenticated key exchange) in Section 20.5. For the remainder of this section we consider the simplest possible attacker. A **passive attacker** or **eavesdropper** (i.e., an attacker who learns g, c_1 and c_2, but does not actively interfere with the protocol) cannot determine K unless they can solve the following computational problem.

Definition 20.2.1. The **Computational Diffie-Hellman problem (CDH)**[1] is: given the triple (g, g^a, g^b) of elements of G to compute g^{ab}.

An extensive discussion of the computational Diffie-Hellman problem will be given in Chapter 21.

Exercise 20.2.2. What is the solution to the CDH instance $(2, 4, 7)$ in the group $\mathbb{F}_{11}^*$?

Suppose one is an eavesdropper on a Diffie-Hellman session and tries to guess the session key K shared by Alice and Bob. The following computational problem is precisely the problem of determining whether the guess for K is correct. This problem arises again later in the chapter in the context of Elgamal encryption.

Definition 20.2.3. Let G be a group and $g \in G$. The **Decisional Diffie-Hellman problem (DDH)** is, given a quadruple (g, g^a, g^b, g^c) of elements in $\langle g \rangle$ to determine whether or not $g^c = g^{ab}$.

Saying that a computational problem such as DDH is hard is slightly less straightforward than with problems like DLP or CDH, since if (g, g^a, g^b, g^c) are chosen uniformly at random in G^4 then the solution to the DDH problem is "no" with overwhelming probability. Clearly, an algorithm that says "no" all the time is not solving the DDH problem,

[1] This assumption comes in two flavours, depending on whether g is fixed or variable. We discuss this issue in more detail later. But, as is the convention in this book, whenever we write "Given...compute..." one should understand that all of the inputs are considered as variables.

so our notion of success must capture this. The correct approach is to define a DDH solver to be an algorithm that can distinguish two distributions on G^4, namely the distribution of **Diffie-Hellman tuples** (i.e., the uniform distribution on tuples of the form $(g, g^a, g^b, g^{ab}) \in G^4$) and the uniform distribution on G^4.

Definition 20.2.4. Let (G_n, r_n) be a family of cyclic groups G_n of order r_n, for $n \in \mathbb{N}$. A **DDH algorithm** for the family G_n is an algorithm A that takes as input a quadruple in G_n^4 and outputs "yes" or "no". The **advantage** of the DDH algorithm A is

$$\begin{aligned}\mathrm{Adv}(A) &= \big|\Pr\left(A\left(g, g^a, g^b, g^{ab}\right) = \text{“}yes\text{”} : g \leftarrow G_n,\ a, b \leftarrow \mathbb{Z}/r\mathbb{Z}\right) \\ &\quad - \Pr\left(A\left(g, g^a, g^b, g^c\right) = \text{“}yes\text{”} : g \leftarrow G_n,\ a, b, c \leftarrow \mathbb{Z}/r\mathbb{Z}\right)\big|.\end{aligned}$$

A DDH algorithm is called **successful** if the advantage is noticeable. The **DDH assumption** for the family of groups is that all polynomial-time (i.e., running time $O(\log(r_n)^c)$ for some constant c) DDH algorithms have negligible advantage.

Lemma 20.2.5. *DDH* $\leq_R$ *CDH* $\leq_R$ *DLP.*

Exercise 20.2.6. Prove Lemma 20.2.5.

Exercise 20.2.7. Definition 20.2.3 states that r is prime. Show that if (g, g^a, g^b, g^c) is a quadruple of elements such that the order of g is n for some integer n where n has some small factors (e.g., factors $l \mid n$ such that $l \leq \log_2(n)$) then one can eliminate some quadruples $(g, g^a, g^b, g^c) \in G^4$ that are not valid DDH tuples by reducing to DDH instances in subgroups of prime order. Show that this is enough to obtain a successful DDH algorithm according to Definition 20.2.4.

20.2.2 Burmester-Desmedt Key Exchange

In the case of $n > 2$ participants there is a generalisation of Diffie-Hellman key exchange due to Burmester and Desmedt [114] that requires two rounds of broadcast. Let the participants in the protocol be numbered as player 0 to player $n-1$. In the first round, player i (for $0 \leq i < n$) chooses a random $1 \leq a_i < r$ and sends $\mathsf{c}_i = g^{a_i}$ to the other players (or, at least, to player $i-1 \pmod n$ and $i+1 \pmod n$). In the second round player i computes

$$\mathsf{t}_i = \left(\mathsf{c}_{i+1 \pmod n}\mathsf{c}_{i-1 \pmod n}^{-1}\right)^{a_i}$$

and sends it to all other players. Finally, player i computes

$$K = \mathsf{c}_{i+1 \pmod n}^{na_i}\mathsf{t}_{i+1 \pmod n}^{n-1}\mathsf{t}_{i+2 \pmod n}^{n-2}\cdots\mathsf{t}_{i+n-1 \pmod n}.$$

Lemma 20.2.8. *Each participant in the Burmester-Desmedt protocol computes*

$$K = g^{a_0a_1 + a_1a_2 + \cdots a_{n-2}a_{n-1} + a_{n-1}a_0}.$$

Exercise 20.2.9. Prove Lemma 20.2.8.

20.2.3 Key Derivation Functions

The result of Diffie-Hellman key exchange is a group element g^{ab}. Typically this should be transformed into an l-bit string for use as a symmetric key (where $l < \log_2(r)$).

Definition 20.2.10. Let G be an algebraic group (or algebraic group quotient) and let l be an integer. A **key derivation function** is a function $\mathsf{kdf} : G \to \{0,1\}^l$. The **output distribution** of a key derivation function is the probability distribution on $\{0,1\}^l$ induced by $\mathsf{kdf}(g)$ over uniformly distributed $g \in G$. A key derivation function is **preimage-resistant** if there is no polynomial-time algorithm known that, on input $x \in \{0,1\}^l$, computes $g \in G$ such that $\mathsf{kdf}(g) = x$.

In general, a key derivation function should have output distribution statistically very close to the uniform distribution on $\{0,1\}^l$. For many applications it is also necessary that kdf be preimage-resistant.

A typical instantiation for kdf is to take a binary representation of $K \in G$, apply a cryptographic hash function (see Chapter 3) to obtain a bit string, and concatenate/truncate as required. See the IEEE P1363 or ANSI X9.42 standards, Section 8 of Cramer and Shoup [160] or Section 6.1 of Raymond and Stiglic [494] for more details; also see Section 3 of [46] for a specific key derivation function for elliptic curves.

20.3 Textbook Elgamal Encryption

In this section we present **textbook Elgamal public key encryption**.[2] This is historically the first public key encryption scheme based on the discrete logarithm problem. As we will see, the scheme has a number of security weaknesses and so is not recommended for practical use. In Chapter 23 we will present secure methods for public key encryption based on computational problems in cyclic groups.

We actually present two "textbook" versions of Elgamal. The first we call "classic textbook Elgamal" as it is essentially the version that appears in [191]. It requires G to be a group (i.e., we cannot use algebraic group quotients) and requires the message m to be encoded as an element of G. Encoding messages as group elements is not difficult, but it is un-natural and inconvenient. The second version, which we call "semi-textbook Elgamal", is more practical as it treats messages as bitstrings. As we will see, the security properties of the two versions are slightly different.

For both schemes, κ denotes a security parameter (so that all attacks should require at least 2^κ bit operations). Figure 20.1 gives classic textbook Elgamal and Figure 20.2 gives semi-textbook Elgamal. We call the sender Bob and the recipient Alice. Messages in the former scheme are group elements and in the latter are l-bit strings, where l depends on the security parameter. Semi-textbook Elgamal also requires a cryptographic hash function $H : G \to \{0,1\}^l$ where G is the group.

Remarks

1. Both versions of textbook Elgamal encryption are best understood as a **static Diffie-Hellman key exchange** followed by symmetric encryption. By this we mean that the sender (Bob) is essentially doing a Diffie-Hellman key exchange with the recipient (Alice): he sends g^k and Alice's component is her fixed (i.e., static) public key g^a. Hence the shared key is g^{ak}, which can then be used as a key for any symmetric encryption scheme (this general approach is known as **hybrid encryption**). The two variants of textbook Elgamal vary in the choice of symmetric encryption scheme: the first uses the map $\mathsf{m} \mapsto \mathsf{m}g^{ak}$ from G to itself while the second uses the map $\mathsf{m} \mapsto \mathsf{m} \oplus H(g^{ak})$ from $\{0,1\}^l$ to itself.

[2] Some authors write "ElGamal" and others write "El Gamal". Reference [191] uses "ElGamal", but we follow the format apparently used nowadays by Elgamal himself.

KeyGen(κ): Run a parameter generation algorithm on security parameter κ that outputs an algebraic group G over a finite field $\mathbb{F}_q$ such that $\#G(\mathbb{F}_q)$ has a prime divisor r and all known algorithms for the discrete logarithm problem in a subgroup of $G(\mathbb{F}_q)$ of order r require at least 2^κ bit operations.

Compute $g \in G$ of prime order r.
Choose a random integer $0 < a < r$ and set $h = g^a$. The public key is (G, g, h) and the private key is a.

The message space is $\mathsf{M}_\kappa = G$.
The ciphertext space is $\mathsf{C}_\kappa = G \times G$.

Encrypt(m): (where $\mathsf{m} \in G$).

- Obtain the public key h of the recipient, Alice.
- Choose a random $0 < k < r$ and set $\mathsf{c}_1 = g^k$.
- Set $\mathsf{c}_2 = \mathsf{m}h^k$.
- Transmit the ciphertext $(\mathsf{c}_1, \mathsf{c}_2)$.

Decrypt($\mathsf{c}_1, \mathsf{c}_2$): Check that $\mathsf{c}_1, \mathsf{c}_2 \in G$. If so, compute and output

$$\mathsf{m} = \mathsf{c}_2\mathsf{c}_1^{-a}.$$

Figure 20.1: Classic Textbook Elgamal Encryption.

2. Elgamal encryption requires two exponentiations in G and decryption requires one. Hence encryption and decryption are polynomial-time and efficient.

3. Elgamal encryption is randomised, so encrypting the same message with the same public key twice will yield two different ciphertexts in general.

4. Unlike RSA, all users in a system can share the same group G. So typically G and g are fixed for all users, and only the value $h = g^a$ changes. Values that are shared by all users are usually called **system parameters**.

20.4 Security of Textbook Elgamal Encryption

We now briefly review the security properties for the textbook Elgamal cryptosystem. First, note that the encryption algorithm should use a good pseudorandom number generator to compute the values for k. A simple attack when this is not the case is given in Exercise 20.4.1.

Exercise 20.4.1. Suppose the random values k used by a signer are generated using the linear congruential generator $k_{i+1} = Ak_i + B \pmod r$ for some $1 \leq A, B < r$. Suppose an adversary knows A and B and sees two classic textbook Elgamal ciphertexts $(\mathsf{c}_1, \mathsf{c}_2)$ and $(\mathsf{c}'_1, \mathsf{c}'_2)$, for the same public key, generated using consecutive outputs k_i and k_{i+1} of the generator. If both ciphertexts are encryptions of the same message then show how the adversary can compute the message. If both ciphertexts are encryptions of different messages then show how to decrypt both ciphertexts using one query to a decryption oracle.

KeyGen(κ): Generate an algebraic group or algebraic group quotient G as in Figure 20.1. Choose a random $g \in G$ of prime order r.
Choose a message size l and a cryptographic hash function $H : G \to \{0,1\}^l$.
Choose a random integer $0 < a < r$ and set $h = g^a$. The public key is (G, H, g, h) and the private key is a.

The message space is $\mathsf{M}_\kappa = \{0,1\}^l$.
The ciphertext space is $\mathsf{C}_\kappa = G \times \{0,1\}^l$.

Encrypt(m): (where $\mathsf{m} \in \{0,1\}^l$).

- Obtain the public key of the recipient, Alice.
- Choose a random $0 < k < r$ and set $\mathsf{c}_1 = g^k$.
- Set $\mathsf{c}_2 = \mathsf{m} \oplus H(h^k)$.
- Transmit the ciphertext $(\mathsf{c}_1, \mathsf{c}_2)$.

Decrypt($\mathsf{c}_1, \mathsf{c}_2$): Check that $\mathsf{c}_1 \in G$ and $\mathsf{c}_2 \in \{0,1\}^l$. If so, compute and output

$$\mathsf{m} = \mathsf{c}_2 \oplus H(\mathsf{c}_1^a).$$

Figure 20.2: Semi-Textbook Elgamal Encryption.

20.4.1 OWE Against Passive Attacks

Theorem 20.4.2. *The computational problem of breaking OWE security of classic textbook Elgamal under passive attack is equivalent to CDH in $\langle g \rangle$.*

Proof: We prove the result only for perfect oracles. To prove OWE-CPA $\leq_R$ CDH, let A be a perfect oracle that solves CDH in the subgroup of order r in G. Call $A(g, h_A, \mathsf{c}_2)$ to get u and set $\mathsf{m} = \mathsf{c}_2 u^{-1}$.

To prove CDH $\leq_R$ OWE-CPA let A be a perfect adversary that takes an Elgamal public key (g, h_A) and an Elgamal ciphertext $(\mathsf{c}_1, \mathsf{c}_2)$ and returns the corresponding message m. We will use this to solve CDH. Let the CDH instance be (g, g_1, g_2). Then choose a random element $\mathsf{c}_2 \in \langle g \rangle$ and call $A(g, g_1, g_2, \mathsf{c}_2)$ to get m. Return $\mathsf{c}_2 \mathsf{m}^{-1}$ as the solution to the CDH instance. □

One can also consider a non-perfect adversary (for example, maybe an adversary can only decrypt some proportion of the possible ciphertexts). It might be possible to develop methods to "self-correct" the adversary using random self-reductions, but this is considered to be the adversary's job. Instead, it is traditional to simply give a formula for the success probability of the algorithm that breaks the computational assumption in terms of the success probability of the adversary. In the context of Theorem 20.4.2, if the adversary can decrypt with noticeable probability ϵ then we obtain a CDH algorithm that is correct with probability ϵ.

Exercise 20.4.3. Prove OWE-CPA $\leq_R$ CDH for semi-textbook Elgamal. Explain why the proof CDH $\leq_R$ OWE-CPA cannot be applied in this case.

20.4.2 OWE Security Under CCA Attacks

We now show that both variants of textbook Elgamal do not have OWE security against an adaptive (CCA) attacker (and hence not IND-CCA security either). Recall that such

an attacker has access to a decryption oracle that will decrypt every ciphertext except the challenge.

Lemma 20.4.4. *Let* $(\mathsf{c}_1, \mathsf{c}_2)$ *be a ciphertext for classic textbook Elgamal with respect to the public key* (G, g, h)*. Suppose* A *is a decryption oracle. Then under a CCA attack one can compute the message corresponding to* $(\mathsf{c}_1, \mathsf{c}_2)$*.*

Proof: Assume that A is perfect. Call A on the ciphertext $(\mathsf{c}_1, \mathsf{c}_2 g) \neq (\mathsf{c}_1, \mathsf{c}_2)$ to obtain a message m'. Then the message corresponding to the original ciphertext is $\mathsf{m} = \mathsf{m}' g^{-1}$.

More generally if A succeeds only with noticeable probability ϵ then we have a CCA2 attack that succeeds with noticeable probability ϵ. □

Another version of this attack follows from Exercises 23.3.3 and 23.3.2.

Exercise 20.4.5. Show that semi-textbook Elgamal encryption does not have the OWE security property under a CCA attack.

We have seen how a CCA attack can lead to an adversary learning the contents of a message. Exercise 20.4.6 gives an example of a general class of attacks called **small subgroup attacks** or **invalid parameter attacks** that can allow a CCA (even a CCA1) adversary to obtain the private key of a user. Such attacks can be performed in many scenarios. One example is when working in a prime order subgroup of $\mathbb{F}_p^*$ where $p-1$ has many small factors. Another example is when using elliptic curves $E : y^2 = x^3 + a_4 x + a_6$; since the addition formula does not feature the value a_6 one can pass an honest user a point of small order on some curve $E'(\mathbb{F}_p)$. A related example is when using x-coordinate only arithmetic on elliptic curves one can choose an x-coordinate corresponding to a point that lies on the quadratic twist. For further discussion is given in Section 4.3 of [273] and a summary of the history of these results is given in Section 4.7 of [273]. We stress that such attacks do not only arise in encryption, but also in authenticated key exchange protocols, undeniable signatures, etc. The general way to avoid such attacks is for all parties to test membership of group elements in every step of the protocol (see Section 11.6).

Exercise 20.4.6. Show how a CCA1 attacker on classic textbook Elgamal can compute u^a for a group element u of their choice where a is the private key of a user. Show that if this attack can be repeated for sufficiently many elements u of coprime small orders then the private key a can be computed.

20.4.3 Semantic Security Under Passive Attacks

A serious problem with the classic textbook Elgamal cryptosystem is that, even though encryption is randomised, it does not necessarily provide semantic security under passive attacks.

Example 20.4.7. Consider the case $G = \mathbb{F}_p^*$, $\mathsf{M} = G$. Let $g \in G$ have prime order r. Then the Legendre symbol of g is $(\frac{g}{p}) = 1$. Hence, the Legendre symbol of the message m satisfies

$$(\tfrac{\mathsf{m}}{p}) = (\tfrac{\mathsf{c}_2}{p})$$

and so can be computed in polynomial-time from the public key and the ciphertext.

To prevent the attack in Example 20.4.7 one can restrict the message space to elements of $\mathbb{F}_p^*$ with Legendre symbol 1. However, this attack is just a special case of a more general phenomenon. The Legendre symbol is a homomorphism $\mathbb{F}_p^* \to G_1$ where $G_1 = \{-1, 1\} \subset \mathbb{F}_p^*$ is the subgroup of order 2. The attack can be performed for any homomorphism onto a subgroup of order coprime to r (this is a slightly different application of the ideas of Section 13.2).

Example 20.4.8. (Boneh, Joux and Nguyen [82]) Let p be a 3072-bit prime and let $r \mid (p-1)$ be a 256-bit prime. Let $g \in \mathbb{F}_p^*$ have order r. Suppose, in violation of the description of classic textbook Elgamal in Section 20.3, one chooses the message space to be

$$\mathsf{M} = \{1, 2, \ldots, 2^{32} - 1\}$$

interpreted as a subset of $\mathbb{F}_p^*$. We identify M with $\{0,1\}^{32} - \{0\}$. Let $(\mathsf{c}_1 = g^k, \mathsf{c}_2 = \mathsf{m}h^k)$ be a challenge ciphertext for classic textbook Elgamal encryption, where $\mathsf{m} \in \mathsf{M}$. Then

$$\mathsf{c}_2^r = \mathsf{m}^r.$$

One expects that, with overwhelming probability, the 2^{32} values m^r are distinct, and hence one can obtain m with at most 2^{32} exponentiations in $\mathbb{F}_p^*$.

Exercise 20.4.9. (Boneh, Joux and Nguyen [82]) Let p and $r \mid (p-1)$ be prime and let $g \in \mathbb{F}_p^*$ have order r. Suppose one uses classic textbook Elgamal with restricted message space $\mathsf{M} = \{0,1\}^m - \{0\}$ as in Example 20.4.8 where $\#\mathsf{M} = 2^m - 1 < p/r$. Extend the attack of Example 20.4.8 using the baby-step-giant-step method, so that it requires $O(2^{m/2+\epsilon})$ exponentiations in G to find m with noticeable probability, for $\epsilon > 0$.

One way to avoid these attacks is to restrict the message space to $\langle g \rangle$. It is then intuitively clear that IND security under passive attacks depends on the decisional Diffie-Hellman problem.

Theorem 20.4.10. *Classic textbook Elgamal with $\mathsf{M} = \langle g \rangle$ has IND-CPA security if and only if the DDH problem is hard.*

Proof: (For perfect oracles.) First we show IND-CPA $\leq_R$ DDH: Let A be an oracle to solve DDH. Let $(\mathsf{c}_1, \mathsf{c}_2)$ be a ciphertext that is an encryption of either m_0 or m_1. Call $A(g, \mathsf{c}_1, h_A, \mathsf{c}_2\mathsf{m}_0^{-1})$ and if the answer is 'yes' then the message is m_0 and if the answer is 'no' then the message is m_1.

For the converse (i.e., DDH $\leq_R$ IND-CPA of Elgamal): Let A be an oracle that breaks indistinguishability of Elgamal. Then A takes as input a public key (g, h), a pair of messages $\mathsf{m}_0, \mathsf{m}_1$ and a ciphertext $(\mathsf{c}_1, \mathsf{c}_2)$ and outputs either 0 or 1. (We assume that A outputs either 0 or 1 even if the ciphertext corresponds to neither message.) Given a DDH instance (g, g_1, g_2, g_3) we repeatedly do the following: choose two random messages m_0 and m_1 in $\langle g \rangle$, choose a random $i \in \{0,1\}$, and call A on the input $(g, g_1, \mathsf{m}_0, \mathsf{m}_1, g_2, \mathsf{m}_i g_3)$. If A outputs i every time then we return 'yes' as the answer to the DDH. If A only outputs the correct answer i about half of the time, then we return 'no'. To be sure the decryption oracle is not just being lucky one should repeat the experiment $\Omega(\log(r))$ times. □

If the hash function is sufficiently good then one does not have to make as strong an assumption as DDH to show that semi-textbook Elgamal encryption has IND security. Instead, the IND security intuitively only depends on CDH. Theorem 20.4.11 is a basic example of a security proof in the random oracle model (see Section 3.7 for background on this model). We give the proof as it illustrates one of the ways the random oracle model is used in theoretical cryptography.

Theorem 20.4.11. *In the random oracle model, semi-textbook Elgamal encryption has IND-CPA security if CDH is hard.*

Proof: (Sketch) Let A be an adversary for the IND-CPA game on semi-textbook Elgamal encryption. Let g, g^a, g^b be a CDH instance. We will describe a simulator S that will solve the CDH problem using A as a subroutine.

First S runs the adversary A with public key (g, g^a).

The simulator must handle the queries made by A to the random oracle. To do this it stores a list of hash values, initially empty. Let g_i be the input for the $i-$th hash query. If $g_i = g_j$ for some $1 \leq j < i$ then we respond with the same value as used earlier. If not then the simulator chooses uniformly at random an element $H_i \in \{0,1\}^l$, stores (g_i, H_i) in the list, and answers the query $H(g_i)$ with H_i. This is a perfect simulation of a random oracle, at least until the challenge ciphertext is issued below.

At some time A outputs a pair of messages m_0 and m_1. The simulator sets $\mathsf{c}_1 = g^b$, chooses c_2 uniformly at random in $\{0,1\}^l$ and responds with the challenge ciphertext $(\mathsf{c}_1, \mathsf{c}_2)$. The adversary A may make further hash function queries (which are answered using the algorithm above) and eventually A outputs $b \in \{0,1\}$ (of course A may crash, or run for longer than its specified running time, in which case S treats this as the output 0).

The logic of the proof is as follows: If A never queries the random oracle H on g^{ab} then A has no information on $H(g^{ab})$ and so cannot determine whether the answer should be 0 or 1. Hence, for A to succeed then one of the queries on H must have been on g^{ab}. Once this query is made then the simulator is seen to be fake as the adversary can check that c_2 is not equal to $\mathsf{m}_b \oplus H(g^{ab})$ for $b \in \{0,1\}$. However, the simulator is not concerned with this issue since it knows that g^{ab} occurs somewhere in the list of hash queries.

The simulator therefore chooses a random index i and responds with g_i as its solution to the CDH instance. □

Exercise 20.4.12. Fill the gaps in the proof of Theorem 20.4.11 and determine the exact probability of success in terms of the success of the adversary and the number of queries to the random oracle.

The power of the random oracle model is clear: we have been able to "look inside" the adversary's computation.

Exercise 20.4.13. Prove the converse to Theorem 20.4.11.

Indeed, the same technique leads to a much stronger result.

Theorem 20.4.14. *In the Random Oracle Model, semi-textbook Elgamal encryption has OWE-CPA security if CDH is hard.*

Exercise 20.4.15. Prove Theorem 20.4.14.

20.5 Security of Diffie-Hellman Key Exchange

A discussion of security models for key exchange is beyond the scope of this book. We refer to Bellare and Rogway [39], Bellare, Pointcheval and Rogaway [37], Bellare, Canetti and Krawczyk [32], Canetti and Krawczyk [116], Shoup [553], Boyd and Mathuria [94] and Menezes, van Oorschot and Vanstone [417] for details. However, as a rough approximation we can consider three types of adversary:

- Passive adversary (also called "benign" in [39]). This attacker obtains all messages sent during executions of the key exchange protocol but does not modify or delete any messages. This attacker is also called an eavesdropper.
- Weak[3] active adversary. This attacker obtains all messages sent during executions of the key exchange protocol and can modify or delete messages. This attacker can also initiate protocol executions with any player.

[3]This use of the word "weak" is non-standard.

- Active adversary. This is as above, but the attacker is allowed to corrupt any honest player who has completed an execution of the protocol and thus obtain the agreed key.

There are two possible goals of an adversary:

- To obtain the shared session key.

- To distinguish the session key from a random key. To make this notion more precise consider a game between an adversary and a challenger. The challenger performs one or more executions of the key exchange protocol and obtains a key K. The challenger also chooses uniformly at random a key K' from the space of possible session keys. The challenger gives the adversary either K or K' (with probability $1/2$). The adversary has to decide whether the received key is K or not. This is called **real or random security**.

The Diffie-Hellman key exchange protocol is vulnerable to a person-in-the-middle attack. Unlike similar attacks on public key encryption, the attacker in this case does not need to replace any users' public keys.

Imagine that an adversary Eve can intercept all communication between Alice and Bob. When Alice sends $\mathsf{c}_1 = g^a$ to Bob, Eve stores c_1 and sends g^e to Bob, for some random integer e known to Eve. Similarly, when Bob sends $\mathsf{c}_2 = g^b$ to Alice, Eve stores c_2 and sends g^e to Alice. Alice computes the key g^{ae} and Bob computes the key g^{be}. Eve can compute both keys. If Alice later sends an encrypted message to Bob using the key g^{ae} then Eve can decrypt it, read it, re-encrypt using the key g^{be}, and forward to Bob. Hence Alice and Bob might never learn that their security has been compromised.

One way to overcome person-in-the-middle attacks is for Alice to send a digital signature on her value g^a (and similarly for Bob). As long as Alice and Bob each hold authentic copies of the other's public keys then this attack fails. Note that this solution does not prevent all attacks on the Diffie-Hellman key exchange protocol.

Another solution is given by authenticated key exchange protocols such as STS, KEA, MTI, MQV, etc (see Chapter 11 of Stinson [591] and the references listed earlier).

We illustrate the basic idea behind most protocols of this type using the **MTI/A0 protocol**: Alice and Bob have public keys $h_A = g^a$ and $h_B = g^b$. We assume that Alice and Bob have authentic copies of each others public keys. They perform Diffie-Hellman key exchange in the usual way (Alice sends g^x and Bob sends g^y). Then the value agreed by both players is

$$g^{ay+bx}.$$

Exercise 20.5.1. Explain why the person-in-the-middle attack fails for this protocol (assuming the public key authentication process is robust).

Exercise 20.5.2. Consider a key exchange protocol where Alice and Bob have public keys $h_A = g^a$ and $h_B = g^b$, where Alice sends g^x and Bob sends g^y and where the shared key is g^{ab+xy}. Show that if corrupt queries are allowed then this key exchange protocol does not provide authentication.

Exercise 20.5.3. Give a person-in-the-middle attack on the Burmester-Desmedt protocol.

20.6 Efficiency Considerations for Discrete Logarithm Cryptography

All cryptographic protocols whose security is related to the DLP involve computations of the form g^a at some stage, and this is usually the most demanding computation in terms of time and computing resources. To make the cryptosystem fast it is natural to try to speed up exponentiation. One could try working in a smaller group, however it is important to ensure that the security of the system is maintained. Indeed, many of the main topics in this book (e.g., tori, elliptic curves and hyperelliptic curves) are attempts to get the "most efficient" group for a given security level.

A number of methods to speed up exponentiation in certain groups have already been presented. Section 11.1 discussed signed expansions, which are suitable for groups (such as elliptic and hyperelliptic curves or tori) where inversion is very efficient. Section 11.3 presented Frobenius expansions and the GLV method, which are suitable for elliptic curves. Those methods all assume that the exponent a takes any value.

One can also consider methods that do not correspond to values a chosen uniformly at random. Such methods can be much faster than the general methods already mentioned, but understanding the security implications can be more complicated. We do not have space to describe any of these methods in detail, but we briefly mention some of them.

1. Choose a to have low Hamming weight. This is mentioned by Agnew, Mullin, Onyszchuk and Vanstone [5] and Schnorr [522].

2. Choose a to be a random Frobenius expansion of low Hamming weight. This is credited to H. W. Lenstra Jr. in Section 6 of Koblitz [346].

3. Choose a to be given by a random addition chain (or addition-subtraction chain). This is proposed in Section 3.3 of Schroeppel, Orman, O'Malley and Spatscheck [531].

4. Choose a to be a product of integers of low Hamming weight. This was proposed and analysed by Hoffstein and Silverman [289].

5. Choosing a to be a random element in GLV representation, possibly with smaller than typical coefficients.

6. Generate random elements using large amounts of precomputation. A solution that can be used in any group is given by Boyko, Peinado and Venkatesan [96]. The method requires precomputing and storing random powers $g_j = g^{a_j}$. One generates a random pair (a, g^a) by taking the product of a random subset of the g^{a_j} and setting $a = \sum a_j \pmod r$. This method is presented as the "simple solution" in [151].

   A more sophisticated method for Koblitz curves is given by Coron, M'Raïhi and Tymen [151]. They use repeated application of sparse Frobenius expansions on elements of the precomputed table. They also give a security analysis.