

Aggelos Kiayias

# Cryptography

## Primitives and Protocols

Notes by S. Pehlivanoglu, J. Todd, and H.S. Zhou

## Contents

# 1 Introduction

To begin discussing the basic properties of cryptography, we consider a simple example.

## 1.1 Flipping a Coin over a Telephone

Suppose Alice and Bob are talking on the phone, debating where they should go for the evening. They agree to toss a coin to see who decides where they go. If Alice and Bob were in the same physical location, they could easily flip a coin and both could verify the result. Since they want to do this over the phone, they need a procedure that enables both parties to verify the outcome and ensures that the outcome is unbiased.

One solution is for Alice to toss a coin into a deep well. This forces Alice to be consistent and prevents her from changing the result. Here the well constitutes a *commitment*. Although Bob still needs to go to the well to check the outcome, by employing the well, both parties no longer need to be physically present to fairly toss a coin.

Since the availability of open wells has decreased in recent years, let us consider a different technique. Assign the value of 1 to Heads and 0 to Tails. Suppose there is a pre-agreed upon mapping  $f$  that sends each of 0 and 1 to a set of objects. In this case,  $f$  plays the role of the well. To determine the outcome of the coin toss,

1. Alice flips a coin and receives  $a \in \{0, 1\}$ . She computes  $f(a)$ .
2. Alice sends  $y = f(a)$  to Bob.
3. Bob flips a coin and receives  $b \in \{0, 1\}$ . He sends  $b$  to Alice.
4. If  $a = b$ , Alice calls Heads; otherwise Alice calls Tails.
5. Alice discloses the value of  $a$  and Bob verifies that  $y$  is a valid commitment to  $a$ .
6. Bob checks if  $a = b$  and confirms the result of Heads or Tails

In order for this protocol to effectively solve the problem,  $f$  must satisfy the following properties:

1. The *hiding property* ensures  $f$  does not reveal any information about  $a$ .
2. The *binding property* requires that it be impossible for Alice to alter the value committed to  $y = f(a)$  and still convince Bob of the validity of the commitment.

If both parties follow the protocol faithfully, the probability distribution of Heads and Tails is uniform for both players; moreover, both parties reach the same conclusion.

Let us now examine what happens if a player deviates from the faithful execution of the protocol. Possible scenarios in which the security of the protocol may be affected include:

1. After obtaining  $b$  in Step 3, Alice substitutes  $a'$  for  $a$  such that  $y = f(a')$ .
2. Bob tries to guess  $a$  after receiving  $y$  and selects  $b$  accordingly.
3. One or both of the players toss their coin in a biased manner such that the probability of Heads or Tails is no longer  $1/2$ .

If  $f$  is chosen accordingly,  $y$  is committed to a certain  $a$  so the binding property prohibits Alice from cheating in the first scenario. Similarly, in the second instance Bob should not be able to effectively guess  $a$  because of the hiding property. The last scenario requires some calculation to determine whether or not different probabilities of  $a$  and  $b$  affect the probability distribution of the players' chances. We have four possibilities.

1. Alice selects  $a = 0$  with probability  $\alpha$ , Bob selects  $b = 0$  with probability  $\beta$ , and the output is Heads;
2. Alice selects  $a = 0$  with probability  $\alpha$ , Bob selects  $b = 1$  with probability  $1 - \beta$ , and the output is Tails;
3. Alice selects  $a = 1$  with probability  $1 - \alpha$ , Bob selects  $b = 0$  with probability  $\beta$ , and the output is Tails;
4. Alice selects  $a = 1$  with probability  $1 - \alpha$ , Bob selects  $b = 1$  with probability  $1 - \beta$ , and the output is Heads.

Then  $\text{Prob}[\text{Heads}] = \alpha\beta + (1 - \alpha)(1 - \beta) = 1 - \alpha - \beta + 2\alpha\beta$ . If both players are dishonest, the protocol will not necessarily function correctly. If one of the parties is honest, so  $\alpha$  or  $\beta = 1/2$ , then  $\text{Prob}[\text{Heads}] = 1/2$ . Based on the above, we can say that the protocol is secure against malicious behavior; it guarantees a uniformly distributed probability to any honest party.

## 1.2 Overview of Cryptography

The previous example illustrates one purpose for cryptography- the need to develop effective commitment schemes. In this class, we will focus on accomplishing five things:

1. Identifying important problems in need of solutions. Here we will see that the previous coin tossing concept is a very important cryptographic protocol with numerous applications in constructing secure systems.
2. Designing solutions for such problems (protocols, algorithms, etc.)
3. Examining the independent components (or primitives) upon which our solutions rely. In Section 1.1 we identified an important primitive in terms of the mapping  $f$ , called a ***bit commitment scheme***.
4. Formally defining security and correctness for all involved parties.
5. Providing proof of security and correctness so as to convince a user that the system satisfies the security and correctness specifications.

In short, we will focus on the goals, designs, primitives, models, and proofs associated with cryptography. The formal, or provable-security approach to the study of cryptography provides mathematical proof that an adversary's objective is either impossible or violates an underlying assumption in a model. An effective solution should satisfy the security model as extensively as possible with the weakest possible assumptions. Those assumptions however, must be plausible.

The provable-security paradigm focuses on two things:

1. Constructing a formal security model and defining what it means for a given cryptographic design to be secure; and
2. Demonstrating that the existence of an adversary capable of efficiently breaking the design's security can be transformed into an algorithm solving a "computationally hard" problem.

The second item introduces an area of interest called ***computational complexity***. This discipline aims to answer questions such as "How many steps are necessary to solve a problem?" or "How much space is required to find a solution to a problem?" One of the objectives of computational complexity is to calculate the time required to find a solution to a problem. For example, one of the fundamental open problems in computer science and mathematics relates to the classes  $P$  and  $NP$ .  $P$  is the set of problems that can be solved in polynomial-time and  $NP$  is the set of problems for

which a candidate solution can be verified in polynomial-time. Although significant effort has been devoted to understanding the relationship between these two classes, it is still unknown if  $P \neq NP$ . It is known however, that a proof of security would imply  $P \neq NP$ . In order to understand this, observe the  $NP$ -nature of cryptography; namely that secret keys play the role of the candidate solutions in a suitable  $NP$  problem. Unfortunately, the fact that  $P \neq NP$  is not helpful in cryptographic security proofs. Such applications ask for average hardness; that is, a random instance of a problem should be computationally hard.

An important tool that assists in the classification of computational problems is the concept of **reduction**. Suppose there are two problems  $A$  and  $B$  and an algorithm  $\alpha$  that solves  $A$  with oracle access to  $B$ , written  $\alpha^B$ . We can appropriately define a pre-order  $\leq^1$  over all problems so  $A \leq B$  if and only if there is an  $\alpha$  where  $\alpha^B$  solves  $A$ . This is a reduction.

Intuitively,  $A \leq B$  implies that  $A$  cannot be substantially harder than  $B$ . Say  $A$  is a well-known hard problem, such as the factoring problem or the discrete logarithm problem, and  $B$  corresponds to breaking the security of our cryptographic construction. If  $A$  is acceptably hard and we can produce a reduction as is specified above, we can assume our construction is provably secure.

Despite the fact that reductions provide little real proof of security, they are acceptable given our general inability to construct a lower bound on the difficulty of computational problems.

## 2 Mathematical Review

Here we give a quick review of algebra, number theory, and probability. Further reviews will be provided as necessary in subsequent sections. <sup>2</sup>

### 2.1 Algebra and Number Theory

#### Groups

**Definition 2.1.1.** A **group**  $(G, *)$  is a set  $G$  together with a binary operation  $*$  satisfying the following conditions:

- $G$  is closed under  $*$ : for all  $g, h \in G, g * h \in G$ ;
- The operation  $*$  is associative: for all  $g, h, \ell \in G, g * (h * \ell) = (g * h) * \ell \in G$ ;
- $G$  contains an identity element  $e$  such that  $g * e = e * g = g$  for all  $g \in G$ ;
- $G$  is closed under inversion: for all  $g \in G$ , there exists  $g^{-1} \in G$  such that  $g * g^{-1} = g^{-1} * g = e$ .

Formally, a group is denoted by an ordered pair  $(G, *)$ . We will write  $G$  when  $*$  is understood.

**Definition 2.1.2.** A group  $G$  is called **Abelian** if for all  $g, h \in G, g * h = h * g$ .

**Theorem 2.1.1.** *If  $G$  is an Abelian group under  $*$ , then  $G$  contains exactly one identity element and every element of  $G$  has a unique inverse.*

**Definition 2.1.3.** In a finite group  $G$ , the **order** of  $G$  is the size or number of elements in the group, denoted  $\#G$  or  $|G|$ .

**Definition 2.1.4.** For a group  $G$  and  $g \in G$ , define the **order of  $g$**  to be the smallest positive integer  $i$  such that  $g^i = e$ , or equivalently,  $\underbrace{g * g * \dots * g}_{i \text{ times}} = e$ . We denote the order of  $g$  by  $\text{ord}(g)$ .

<sup>1</sup>A **pre-order** is a reflexive, transitive relation.

<sup>2</sup>For more mathematical review, see [?].

**Theorem 2.1.2 (Lagrange).** *In a finite group, the order of any element divides the size of the group.*

**Definition 2.1.5.** If there is some element  $g \in G$  such that  $\text{ord}(g) = \#G$ , then  $g$  generates  $G$  and we call  $G$  a **cyclic group**. We write  $G = \langle g \rangle$ .

**Example.** Consider  $\mathbb{Z}_5^* = \mathbb{Z}_5 - \{0\}$ . This is a cyclic group under multiplication modulo 5. Our goal is to find  $g \in \mathbb{Z}_5^*$  such that  $\text{ord}(g) = \#\mathbb{Z}_5^* = 4$  and therefore  $\langle g \rangle = \mathbb{Z}_5^*$ . Clearly  $\langle 1 \rangle \neq \mathbb{Z}_5^*$ , so let us try 2:

$$2^0 \equiv 1 \pmod{5}, \quad 2^1 \equiv 2 \pmod{5}, \quad 2^2 \equiv 4 \pmod{5}, \quad 2^3 \equiv 3 \pmod{5}, \quad \text{and} \quad 2^4 \equiv 1 \pmod{5}.$$

Since  $\langle 2 \rangle = \{1, 2, 3, 4\}$  and 2 has order 4, 2 generates  $\mathbb{Z}_5^*$ .

It is possible for multiple elements to generate the group, so let us now try 3. By Lagrange,  $\text{ord}(3) \mid 4$ . From our previous calculations,  $2^3 \equiv 3 \pmod{5}$ , so  $\text{ord}(2^3) = \text{ord}(3)$ . Then  $3^{\text{ord}(3)} \equiv 2^{3 \cdot \text{ord}(3)} \equiv 1 \pmod{5}$  and  $3 \cdot \text{ord}(3) \equiv \text{ord}(2) \pmod{4}$ . Since 3 and 4 are relatively prime,  $\text{ord}(3) = 4$ . Thus 3 is another generator of  $\mathbb{Z}_5^*$ .

By the same argument, we can show 4 is not a generator. From the above,  $2^2 \equiv 4 \pmod{5}$ , so  $\text{ord}(2^2) = \text{ord}(4)$ . We know that 2 (the exponent in  $2^2$ ) divides 4, therefore  $\text{gcd}(2, 4)$  divides 4. Moreover,  $\text{ord}(4) = 4 / \text{gcd}(2, 4) = 2$ . This implies  $\#\langle 4 \rangle = 2$ , so 4 is not a generator:  $\langle 4 \rangle = \{1, 4\}$ .

## Rings and Fields

**Definition 2.1.6.** A (**commutative**) **ring**  $R$  is a set together with two binary operations  $+$  and  $*$  such that

- $(R, +)$  is an Abelian group;
- The operation  $*$  is associative:  $(r * s) * t = r * (s * t)$  for all  $r, s, t \in R$ ;
- The distributive law holds in  $R$ :  $r * (s + t) = r * s + r * t$  and  $(r + s) * t = r * t + s * t$  for all  $r, s, t \in R$ ;
- The operation  $*$  commutes:  $r * s = s * r$  for all  $r, s \in R$ ; and
- $R$  contains an identity if there is an element  $1 \in R$  such that  $1 * r = r * 1 = r$  for all  $r \in R$ .

Simply put, a commutative ring is an Abelian group without inverses. Not all rings contain 1, so the last condition is not absolute.

**Example.**  $\mathbb{Z}$  is a ring under the usual addition and multiplication.

**Example.**  $\mathbb{Z}_n$  is a ring under addition and multiplication modulo  $n$ .

**Definition 2.1.7.** A **field**  $F$  is a set together with two binary operations  $+$  and  $*$  such that

- $(F, +)$  is an Abelian group with identity 0;
- $(F - \{0\}, *)$  is an Abelian group with identity 1 and the distributive law holds.

**Example.**  $\mathbb{Q}, \mathbb{R}$ , and  $\mathbb{C}$  are all fields under the usual addition and multiplication.

**Example.** For any prime  $p$ ,  $\mathbb{Z}_p$  is a field under addition and multiplication modulo  $p$ .

**Definition 2.1.8.** Let  $p$  be a prime. Then  $\mathbb{Z}_p$  is a **finite field**, denoted  $\mathbb{F}_p$ .

### Chinese Remainder Theorem

**Definition 2.1.9.** We denote *congruence relationships* over the integers by  $a \equiv b \pmod n$  if and only if  $n \mid (a - b)$ .

**Theorem 2.1.3 (Chinese Remainder Theorem).** Let  $m_1, \dots, m_k$  be pairwise relatively prime positive integers and let  $c_1, \dots, c_k$  be arbitrary integers. Then there exists an integer  $x$  such that

$$x \equiv c_i \pmod{m_i}$$

for all  $i = 1, \dots, k$ . Moreover, any integer  $x'$  is also a solution to these congruences if and only if  $x \equiv x' \pmod M$  where  $M = \prod m_i$  for  $i = 1, \dots, k$ .

*Proof.* Let  $M = \prod m_i$  for  $i = 1, \dots, k$ . Define  $m'_i = M/m_i$ . All the  $m_i$ s are pairwise relatively prime, so  $\gcd(m_i, m'_i) = 1$  for all  $i$ . Let  $u_i = (m'_i)^{-1} \pmod{m_i}$  and  $w_i = m'_i u_i$ . By construction then,  $w_i \equiv 1 \pmod{m_i}$  and  $w_i \equiv 0 \pmod{m_j}$  when  $i \neq j$ . This gives us  $w_i \equiv \delta_{ij} \pmod{m_j}$  where

$$\delta_{ij} = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{if } i \neq j. \end{cases}$$

Letting  $x = \sum w_i c_i$  for  $i = 1, \dots, k$ , we see

$$x \equiv \sum_{j=1}^k \delta_{ij} c_i \equiv c_j \pmod{m_j}$$

as desired. ■

**Remark.** The Chinese Remainder Theorem implies the group isomorphism

$$\mathbb{Z}_n^* \cong \mathbb{Z}_{p_1^{e_1}}^* \times \dots \times \mathbb{Z}_{p_m^{e_m}}^*,$$

given by  $a \pmod n \mapsto (a \pmod{p_1^{e_1}}, \dots, a \pmod{p_m^{e_m}})$ , where  $n = p_1^{e_1} \dots p_m^{e_m}$  for integers  $e_i$  and distinct primes  $p_i$ .

**Example.** Historically, the Chinese used this theorem to count soldiers. After a battle, the soldiers would line up in rows of (for example) three, then in rows of five, and then in rows of seven. By counting the remaining soldiers after each formation, the commanders could quickly determine the total number of men and therefore determine their losses.

Say there are fewer than 100 soldiers. After lining up 3 soldiers in each row, 1 soldier remains. After standing 5 in a row, 2 soldiers remain, and after standing 7 in a row, 6 remain. We want to calculate the exact number of soldiers.

Let  $x$  represent the total. Then

$$\begin{aligned} x &\equiv 1 \pmod 3 \\ x &\equiv 2 \pmod 5 \\ x &\equiv 6 \pmod 7. \end{aligned}$$

We compute  $M = 3 \cdot 5 \cdot 7 = 105$ , and  $m'_1 = 35$ ,  $m'_2 = 21$ ,  $m'_3 = 15$ . Computing inverses now, we have

$$\begin{aligned} u_1 &= 35^{-1} \equiv 2 \pmod 3 \\ u_2 &= 21^{-1} \equiv 1 \pmod 5 \\ u_3 &= 15^{-1} \equiv 1 \pmod 7 \end{aligned}$$

Then  $w_1 = 70$ ,  $w_2 = 21$ ,  $w_3 = 15$ , making  $x = w_1 c_1 + w_2 c_2 + w_3 c_3 = 70(1) + 21(2) + 15(6) \equiv 97 \pmod{105}$ . Thus there are 97 soldiers.

## 2.2 Discrete Probability

**Definition 2.2.1.** A *discrete probability distribution*  $\mathbf{D}$  over a set  $[\mathbf{D}]$  is specified as

- $\text{Prob}_{\mathbf{D}}[u] \in [0, 1]$  for all  $u \in [\mathbf{D}]$
- $\sum_{u \in \mathbf{D}} \text{Prob}_{\mathbf{D}}[u] = 1$ .

The set  $[\mathbf{D}]$  is called the *support* of  $\mathbf{D}$ .

**Example (Succeeding by Repetition).** Consider an experiment where the probability of success is  $p$ . Suppose the experiment is repeated  $n$  times; we want to bound the probability that all  $n$  trials fail. Since each trial is independent of the next, the probability of  $n$  failures is  $(1 - p)^n$ . Recall that  $1 - x \leq e^{-x}$  for all  $x$ . By letting  $x = p$  and raising both sides to the  $n$ th power, we obtain the upper bound  $(1 - p)^n \leq e^{-pn}$ . Then

$$\begin{aligned} \text{Prob}[\text{At least 1 success}] &= 1 - \text{Prob}[\text{All fail}] \\ &\geq 1 - e^{-pn}. \end{aligned}$$

If  $p$  is fixed, the probability that all trials fail drops exponentially according to the number of repetitions  $n$ .

**Example (Balls and Boxes).** Consider an experiment with  $n$  boxes and  $k$  balls, each of a different color. Each ball is thrown into a box at random. We define a collision to be the event that 2 different colored balls land in the same box. We want to calculate the probability of a collision. In a situation like this, it is often easier to calculate the probability of the complementary event:

$$\text{Prob}_{\mathbf{D}}[\text{No collision}] = \frac{n(n-1) \cdots (n-k+1)}{n^k} = \prod_{j=0}^{k-1} \left( \frac{n-j}{n} \right).$$

Using again the fact that  $1 - x \leq e^{-x}$  for all  $x$ , we have

$$\prod_{j=0}^{k-1} \left( \frac{n-j}{n} \right) = \prod_{j=1}^{k-1} \left( 1 - \frac{j}{n} \right) \leq \prod_{j=1}^{k-1} e^{-j/n} = e^{-k(k-1)/2n}.$$

Since  $\text{Prob}[\text{Collision}] = 1 - \text{Prob}[\text{No collision}]$ ,

$$\text{Prob}_{\mathbf{D}}[\text{Collision}] \geq 1 - e^{-k(k-1)/2n}.$$

**Example (The Birthday Paradox).** The Birthday Paradox is a classic problem utilizing the previous scheme. We want to know how many people must be in a room for there to be at least a 50% chance that two people have the same birthday (a collision). Let  $n = 365$  and assume that people's birthdays are uniformly distributed over the days of the year. If we want  $\text{Prob}_{\mathbf{D}}[\text{Collision}] \geq 1/2$ , then

$$\begin{aligned} 1 - e^{-k(k-1)/2n} &\geq \frac{1}{2} \\ e^{-k(k-1)/2n} &\leq \frac{1}{2} \\ e^{k(k-1)/2n} &\geq 2 \\ \frac{k(k-1)}{2n} &\geq \ln 2 \\ \frac{k^2}{2n} &\geq \ln 2 \\ k &\geq \sqrt{2n \ln 2} \end{aligned}$$



So if there are more than 23 people in a room, there is over a 50% chance that two people share the same birthday. This seems a bit counterintuitive; hence the name “paradox”.

**Example (Binomial Distribution).** A *binomial trial* is an experiment with only two possible outcomes: success and failure. Let  $[\mathbf{D}] = \{0, 1, \dots, n\}$  and the probability of one success be  $p$ , then the *binomial distribution* is the probability of  $u$  successes in a sequence of  $n$  independent trials:

$$\text{Prob}_{\mathbf{D}}[u] = \binom{n}{u} p^u (1-p)^{n-u}.$$

**Definition 2.2.2.** A subset  $\mathbf{A} \subseteq [\mathbf{D}]$  denotes an *event*. The probability of  $\mathbf{A}$  is

$$\text{Prob}_{\mathbf{D}}[\mathbf{A}] = \sum_{u \in \mathbf{A}} \text{Prob}_{\mathbf{D}}[u].$$

It is also possible to prove various statements about set theoretical operations defined between events, such as unions and intersections. For example, if  $\mathbf{A}, \mathbf{B} \subseteq [\mathbf{D}]$ , we have

$$\text{Prob}_{\mathbf{D}}[\mathbf{A} \cup \mathbf{B}] = \text{Prob}_{\mathbf{D}}[\mathbf{A}] + \text{Prob}_{\mathbf{D}}[\mathbf{B}] - \text{Prob}_{\mathbf{D}}[\mathbf{A} \cap \mathbf{B}].$$

This is called the *inclusion-exclusion principal*.

## 2.3 Conditional Probability

**Definition 2.3.1.** Let  $\mathbf{A}$  and  $\mathbf{B}$  be two events. The probability of  $\mathbf{A}$  occurring, given that  $\mathbf{B}$  has already occurred is called a *conditional probability*. This is given by

$$\text{Prob}_{\mathbf{D}}[\mathbf{A} \mid \mathbf{B}] = \frac{\text{Prob}_{\mathbf{D}}[\mathbf{A} \cap \mathbf{B}]}{\text{Prob}_{\mathbf{D}}[\mathbf{B}]}.$$

The following theorem is useful for computing conditional probabilities.

**Theorem 2.3.1 (Bayes).** For two events  $\mathbf{A}$  and  $\mathbf{B}$ ,

$$\text{Prob}_{\mathbf{D}}[\mathbf{B} \mid \mathbf{A}] = \frac{\text{Prob}_{\mathbf{D}}[\mathbf{A} \mid \mathbf{B}] \cdot \text{Prob}_{\mathbf{D}}[\mathbf{B}]}{\text{Prob}_{\mathbf{D}}[\mathbf{A}]}.$$

Moreover, if  $\mathbf{D}_1, \dots, \mathbf{D}_n$  is a partition of disjoint events of  $[\mathbf{D}]$  such that  $[\mathbf{D}] = \bigcup \mathbf{D}_i$  for  $1 \leq i \leq n$ , then for any events  $\mathbf{A}$  and  $\mathbf{B}$ ,

$$\text{Prob}_{\mathbf{D}}[\mathbf{B} \mid \mathbf{A}] = \frac{\text{Prob}_{\mathbf{D}}[\mathbf{A} \mid \mathbf{B}] \cdot \text{Prob}_{\mathbf{D}}[\mathbf{B}]}{\sum_{i=1}^n \text{Prob}_{\mathbf{D}}[\mathbf{A} \mid \mathbf{D}_i] \cdot \text{Prob}_{\mathbf{D}}[\mathbf{D}_i]}.$$

Let  $\overline{\mathbf{B}}$  denote the complement of an event:  $\overline{\mathbf{B}} = [\mathbf{D}] \setminus \mathbf{B}$ . Bayes' theorem suggests

$$\text{Prob}_{\mathbf{D}}[\mathbf{B} \mid \mathbf{A}] = \frac{\text{Prob}_{\mathbf{D}}[\mathbf{A} \mid \mathbf{B}] \cdot \text{Prob}_{\mathbf{D}}[\mathbf{B}]}{\text{Prob}_{\mathbf{D}}[\mathbf{A} \mid \mathbf{B}] \cdot \text{Prob}_{\mathbf{D}}[\mathbf{B}] + \text{Prob}_{\mathbf{D}}[\mathbf{A} \mid \overline{\mathbf{B}}] \cdot \text{Prob}_{\mathbf{D}}[\overline{\mathbf{B}}]}.$$

**Example.** Here we see an application of Bayes' Theorem. Let  $\mathbf{D}$  be a probability distribution over a given population and let the event  $\mathbf{S}$  correspond to the subset of the population sick with a certain disease. Suppose there is a medical test that checks for the disease and define  $\mathbf{T}$  to be the event that an individual selected from the population tests positive.

The prevalence of the disease is  $\text{Prob}_{\mathbf{D}}[\mathbf{S}] = 1\%$ , the chances of a successful test are  $\text{Prob}_{\mathbf{D}}[\mathbf{T} \mid \mathbf{S}] = 99\%$ , and the probability of an inaccurate test is  $\text{Prob}_{\mathbf{D}}[\mathbf{T} \mid \overline{\mathbf{S}}] = 5\%$ . We want to find the probability that a certain individual is sick, given that the test result is positive. A common

mistake is to claim that the probability is 99%- the success rate of the test. This is false because it fails to take into account that we already know the person tested positive. Using Bayes' theorem, we can account for this information and compute

$$\begin{aligned} \text{Prob}_{\mathbf{D}}[\mathbf{S} \mid \mathbf{T}] &= \frac{\text{Prob}_{\mathbf{D}}[\mathbf{T} \mid \mathbf{S}] \cdot \text{Prob}_{\mathbf{D}}[\mathbf{S}]}{\text{Prob}_{\mathbf{D}}[\mathbf{T} \mid \mathbf{S}] \cdot \text{Prob}_{\mathbf{D}}[\mathbf{S}] + \text{Prob}_{\mathbf{D}}[\mathbf{T} \mid \overline{\mathbf{S}}] \cdot \text{Prob}_{\mathbf{D}}[\overline{\mathbf{S}}]} \\ &= \frac{(0.99)(0.01)}{(0.99)(0.01) + (0.05)(1 - 0.01)} \\ &= \frac{1}{6}. \end{aligned}$$

This might seem unreasonable, but because the disease is so uncommon, a positive test is more likely to occur from an inaccurate test than from the actual sickness.

## 2.4 Random Variables

**Definition 2.4.1.** For a probability distribution  $\mathbf{D}$ , we define a *random variable*  $X$  to be a function

$X: [\mathbf{D}] \rightarrow R$ . For any  $x \in R$ , we use the notation

$$\text{Prob}[X = x] = \sum_{X(u)=x} \text{Prob}_{\mathbf{D}}[u].$$

We say that a random variable  $X$  is distributed according to  $\mathbf{D}$  if  $X: [\mathbf{D}] \rightarrow [\mathbf{D}]$  is the identity function. We denote this by

$$\text{Prob}_{X \leftarrow \mathbf{D}}[X = x] = \sum_{X(u)=x} \text{Prob}_{\mathbf{D}}[u].$$

**Definition 2.4.2.** For any probability distribution  $\mathbf{D}$  with random variable  $X$ , its *expectation* is

$$\text{E}[X] = \sum_{x \in R} x \text{Prob}[X = x].$$

**Definition 2.4.3.** The *variance* of a discrete random variable  $X$  measures the spread, or variability of a distribution. It is defined by

$$\text{Var}[X] = \text{E}[X^2] - \text{E}[X]^2.$$

## 2.5 Tails of Probability Distributions

When analyzing random procedures, we often want to estimate the bounds on the *tails* of a probability distribution. The term “tails” refers to the extremities of the graphical representation of a probability distribution, where the distribution deviates from the mean. The following theorems will be helpful.

**Theorem 2.5.1 (Markov's Inequality).** *Let  $X$  be a random variable that takes only nonnegative real values. Then for any  $t > 0$ ,*

$$\text{Prob}[X \geq t] \leq \frac{\text{E}[X]}{t}.$$

**Theorem 2.5.2 (Chebyshev's Inequality).** *Let  $X$  be a random variable. For any  $t > 0$  we have*

$$\text{Prob}[|X - \text{E}(X)| \geq t] \leq \frac{\text{Var}[X]}{t^2}.$$

**Theorem 2.5.3 (Chernoff's Bound).** Let  $X_1, \dots, X_n$  be independent random variables taking values in  $\{0, 1\}$  with  $\text{Prob}[X_i = 1] = p_i$ . Then

$$\text{Prob}\left[\sum_{i=1}^n X_i \leq (1 - \delta)\mu\right] \leq e^{-\mu\delta^2/2} \quad \text{and} \quad \text{Prob}\left[\sum_{i=1}^n X_i \geq (1 + \delta)\mu\right] \leq \left(\frac{e^\delta}{(1 + \delta)^{1+\delta}}\right)^\mu$$

where  $\mu = \sum p_i$  and  $\delta \in (0, 1]$ .

Here  $\mu$  is the expectation and  $(1 - \delta)\mu$  and  $(1 + \delta)\mu$  are the tails.

**Example (Guessing with a Majority).** Suppose there is an oracle that answers questions with Yes or No, and answers questions correctly with probability  $1/2 + \alpha$ . Say we ask the oracle  $n$  questions and let  $X_i$  be a random variable according to

$$X_i = \begin{cases} 1, & \text{oracle answers the } i\text{th query correctly} \\ 0, & \text{otherwise.} \end{cases}$$

If we define a failure as receiving fewer correct answers than incorrect answers, the probability of failing is

$$\text{Prob}[\text{Failure}] = \text{Prob}\left[\# \text{ of correct answers} \leq \frac{n}{2}\right] = \text{Prob}\left[\sum_{i=1}^n X_i \leq \frac{n}{2}\right].$$

Here we apply Chernoff's bound by setting  $n/2 = (1 - \delta)\mu$ . Then

$$\text{Prob}\left[\sum_{i=1}^n X_i \leq (1 - \delta)\mu\right] \leq e^{-\mu\delta^2/2}. \quad (1)$$

Noting that  $\mu = (1/2 + \alpha)n$ , we can solve for  $\delta$ .

$$\begin{aligned} \frac{n}{2} &= (1 - \delta)\mu \\ \frac{n}{2} &= (1 - \delta)\left(\frac{1}{2} + \alpha\right)n \\ \delta &= \frac{\alpha}{1/2 + \alpha} \end{aligned}$$

To estimate the probability of a failure, we substitute this value of  $\delta$  into (1).

$$\text{Prob}[\text{Failure}] \leq e^{-\alpha^2 n / (1 + 2\alpha)}.$$

This implies that if the oracle has bias  $\alpha$ , we can typically expose the bias after a sufficient number of repetitions  $n$ . Because of this, the probability of failing drops exponentially depending on the degree of the bias and the number of trials. If we want the probability of failing to fall below some  $\varepsilon$ , we can find a suitable lower bound on  $n$ .

$$\begin{aligned} e^{-\alpha^2 n / (1 + 2\alpha)} &< \varepsilon \\ \frac{-\alpha^2 n}{(1 + 2\alpha)} &< \ln(\varepsilon) \\ n &> \alpha^{-2}(1 + 2\alpha) \ln\left(\frac{1}{\varepsilon}\right) \end{aligned}$$

So by taking  $n$  large enough, we can guarantee that the probability of failing is sufficiently low.

## 2.6 Statistical Distance

**Definition 2.6.1.** Let  $X$  and  $Y$  be random variables distributed according to  $\mathbf{D}_1$  and  $\mathbf{D}_2$  respectively and let  $\mathbf{V} = X([\mathbf{D}_1]) \cup Y([\mathbf{D}_2])$ . We define the *statistical distance*  $\Delta$  by

$$\Delta[X, Y] = \frac{1}{2} \sum_{u \in \mathbf{V}} \left| \text{Prob}_{X \leftarrow \mathbf{D}_1} [X = u] - \text{Prob}_{Y \leftarrow \mathbf{D}_2} [Y = u] \right|.$$

Figure 1 illustrates the statistical distance between two random variables  $X$  and  $Y$ . The dotted curve represents the distribution of  $X$  over  $\mathbf{D}_1$  and the black curve corresponds to  $Y$  over  $\mathbf{D}_2$ . By definition, the sum of the probabilities over the support set is 1, so the area below each curve is 1. Half the sum of the shaded areas represents the statistical distance between  $X$  and  $Y$ . Because the striped area equals the gray area, dividing the total shaded area by 2 effectively establishes one of the two marked areas as the statistical distance.

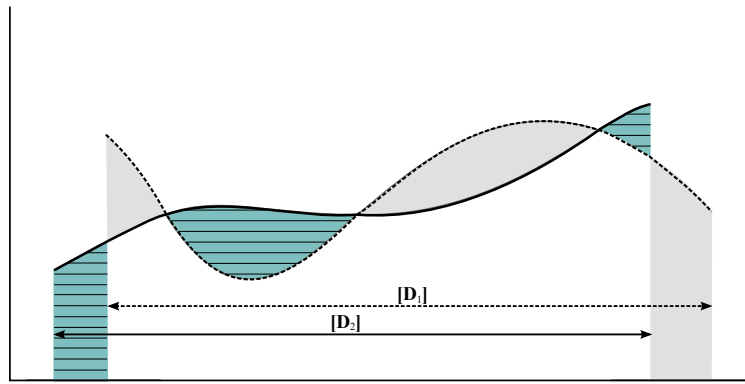


Figure 1: Two probability distributions over different support sets  $[\mathbf{D}_1]$  and  $[\mathbf{D}_2]$ . The shaded regions distinguish the statistical distance between the random variables.

*Exercise:* Show that for any two support sets  $[\mathbf{D}_1]$  and  $[\mathbf{D}_2]$ , the striped area equals the gray area, so the statistical distance is equal to one of the two areas.

**Definition 2.6.2.** Let  $\varepsilon > 0$ , then two random variables  $X$  and  $Y$  are said to be  $\varepsilon$ -*close* if  $\Delta[X, Y] \leq \varepsilon$ .

**Example.** Let  $\mathbf{D}_1$  be the uniform distribution over  $[0, A)$  where  $2^n \leq A < 2^{n+1}$  and let  $\mathbf{D}_2$  be the uniform distribution over  $[0, 2^n)$ . We want to calculate the statistical distance of  $\mathbf{D}_1$  and  $\mathbf{D}_2$ .

Since  $\mathbf{D}_1$  is uniform over  $[0, A)$ , we have  $\text{Prob}_{\mathbf{D}_1}[u] = 1/A$  for all  $u \in [0, A)$ . Similarly, we can extend  $\mathbf{D}_2$  over the sample space  $[0, A)$  by defining

$$\text{Prob}_{\mathbf{D}_2}[u] = \begin{cases} 1/2^n, & u \in [0, 2^n) \\ 0, & u \in [2^n, A). \end{cases}$$

Suppose  $X$  and  $Y$  are random variables distributed according to  $\mathbf{D}_1$  and  $\mathbf{D}_2$  respectively where

$[\mathbf{D}_1] = [\mathbf{D}_2] = [0, A]$ . Then

$$\begin{aligned} \Delta[X, Y] &= \frac{1}{2} \sum_{u \in [0, A]} \left| \text{Prob}_{X \leftarrow \mathbf{D}_1}[X = u] - \text{Prob}_{X \leftarrow \mathbf{D}_2}[X = u] \right| \\ &= \frac{1}{2} \left( \sum_{u \in [0, 2^n)} \left| \frac{1}{A} - \frac{1}{2^n} \right| + \sum_{u \in [2^n, A)} \left| \frac{1}{A} - 0 \right| \right) \\ &= \frac{1}{2} \left( \sum_{u \in [0, 2^n)} \left( \frac{1}{2^n} - \frac{1}{A} \right) + \sum_{u \in [2^n, A)} \frac{1}{A} \right) \\ &= \frac{1}{2} \left( \left( \frac{1}{2^n} - \frac{1}{A} \right) 2^n + \frac{1}{A} (A - 2^n) \right) \\ &= \frac{A - 2^n}{A}. \end{aligned}$$

Letting  $d = A - 2^n$ , we have  $\Delta[X, Y] = d/(d + 2^n)$ . When  $A$  is relatively close to  $2^n$ ,  $\Delta[X, Y]$  approximates 0. For example, if  $d = 2^{n/2}$  so that  $A = 2^{n/2} + 2^n$ , the statistical distance drops exponentially:

$$\frac{d}{d + 2^n} = \frac{2^{n/2}}{2^{n/2} + 2^n} = \frac{1}{1 + 2^{n/2}} \approx 2^{-n/2}.$$

**Definition 2.6.3.** A function  $f$  is *negligible* if for all  $c \in \mathbb{R}$  there exists  $n_0 \in \mathbb{N}$  such that  $f(n) \leq 1/n^c$  for all  $n \geq n_0$ .

**Definition 2.6.4.** A (*probability*) *ensemble* is a collection of distributions  $\mathcal{D} = \{\mathbf{D}_n\}_{n \in \mathbb{N}}$ .

We now take the collection  $X$  over an ensemble  $\mathcal{D}$  to mean a collection of random variables over  $\mathbf{D}_n \in \mathcal{D}$ . As an abuse of notation however, we will still refer to the collection  $X$  as a random variable.

**Definition 2.6.5.** Let  $X$  and  $Y$  be random variables over ensembles  $\mathcal{D}$  and  $\mathcal{D}'$ . We say  $\mathcal{D}$  and  $\mathcal{D}'$  are *statistically indistinguishable* if  $\Delta[X, Y]$  is a negligible function in  $n$ .

It needs to be stressed that  $\Delta[X, Y] \rightarrow 0$  for two ensembles does not imply that the ensembles are indistinguishable. Statistical indistinguishability implies that the statistical distance, when viewed as a function of  $n$ , should be smaller than any polynomial function of  $n$  for sufficiently large values of  $n$ .

## 2.7 An Alternate Definition of Statistical Distance

**Definition 2.7.1.** A *statistical test*  $\mathcal{A}$  for an ensemble  $\mathcal{D} = \{\mathbf{D}_n\}_{n \in \mathbb{N}}$  is an algorithm that takes input elements from  $\mathbf{D}_n$  and outputs values in  $\{0, 1\}$  for each  $n \in \mathbb{N}$ .

**Theorem 2.7.1.** Consider the statistical test  $\mathcal{A}$  as a function of  $n$  and let  $X$  and  $Y$  be random variables following the ensembles  $\mathcal{D}_1$  and  $\mathcal{D}_2$  respectively. Define

$$\Delta_{\mathcal{A}}[X, Y] = \left| \text{Prob}_{X \leftarrow \mathcal{D}_1}[\mathcal{A}(X) = 1] - \text{Prob}_{Y \leftarrow \mathcal{D}_2}[\mathcal{A}(Y) = 1] \right|$$

to be the *statistical distance* with respect to the test  $\mathcal{A}$ . Then for all  $\mathcal{A}$ ,  $\Delta[X, Y] \geq \Delta_{\mathcal{A}}[X, Y]$  and there exists some  $\mathcal{A}^*$  such that  $\Delta[X, Y] = \Delta_{\mathcal{A}^*}[X, Y]$ .

The first part of the theorem is argued as follows. For any  $\mathcal{A}$ ,

$$\Delta_{\mathcal{A}}[X, Y] = \left| \sum_{a \in A_n} \text{Prob}_{\mathbf{D}_1}[a] - \text{Prob}_{\mathbf{D}_2}[a] \right| \leq \sum_{a \in A_n} |\text{Prob}_{\mathbf{D}_1}[a] - \text{Prob}_{\mathbf{D}_2}[a]| =_{\text{df}} N_1$$

where  $A_n = \{a \in \mathbf{D}_n : \mathcal{A}(a) = 1\}$ .

Now consider the statistical test  $\overline{\mathcal{A}}$  that operates exactly as  $\mathcal{A}$  but flips the answer. It immediate that  $\Delta_{\overline{\mathcal{A}}}[X, Y] = \Delta_{\mathcal{A}}[X, Y]$  based on the definition of  $\Delta_{\mathcal{A}}[\cdot, \cdot]$ . Based on a similar reasoning as above we have that

$$\Delta_{\mathcal{A}}[X, Y] = \Delta_{\overline{\mathcal{A}}}[X, Y] \leq \sum_{a \in \overline{A_n}} |\text{Prob}_{\mathbf{D}_1}[a] - \text{Prob}_{\mathbf{D}_2}[a]| =_{\text{df}} N_2$$

where  $\overline{A_n}$  is the complement of  $A_n$  in  $\mathbf{D}_n$ .

Now we observe that  $N_1 + N_2 = \Delta[X, Y]$  and given that  $N_1, N_2 \in [0, 1]$  it holds that one of them is at most  $\frac{1}{2}\Delta[X, Y]$ . The result follows.

Regarding the second part of the theorem, we define a distinguisher  $\mathcal{A}^*$  as follows:

$$\mathcal{A}^*(a) = \begin{cases} 1, & \text{Prob}_{\mathbf{D}_1}[a] \geq \text{Prob}_{\mathbf{D}_2}[a] \\ 0, & \text{otherwise,} \end{cases}$$

it follows easily that  $\Delta[X, Y] = \Delta_{\mathcal{A}^*}[X, Y]$ . Indeed, for  $A_n^* = \{a \in \mathbf{D}_n : \mathcal{A}^*(a) = 1\} \subseteq \mathbf{D}_n$ ,

$$\Delta_{\mathcal{A}^*}[X, Y] = \left| \sum_{a \in A_n^*} \text{Prob}_{\mathbf{D}_1}[a] - \sum_{a \in A_n^*} \text{Prob}_{\mathbf{D}_2}[a] \right| = \sum_{a \in A_n^*} (\text{Prob}_{\mathbf{D}_1}[a] - \text{Prob}_{\mathbf{D}_2}[a])$$

from which the result follows immediately.

To visualize how  $\Delta[X, Y] = \Delta_{\mathcal{A}^*}[X, Y]$  for this distinguisher, we return to Figure 1. The striped area denotes where  $\text{Prob}_{\mathbf{D}_1}[u] \geq \text{Prob}_{\mathbf{D}_2}[u]$ , which we have already seen is exactly the statistical distance.

**Example.** Consider the two probability distributions  $\mathbf{D}_1$  and  $\mathbf{D}_2$  where

$\mathbf{b}_1 \mathbf{b}_0$	$\mathbf{D}_1$	$\mathbf{D}_2$
0 0	$0.25 - \varepsilon$	0.25
0 1	0.25	0.25
1 0	$0.25 + \varepsilon$	0.25
1 1	0.25	0.25

Let  $X$  and  $Y$  be random variables following  $\mathbf{D}_1$  and  $\mathbf{D}_2$ . The statistical distance is

$$\Delta[X, Y] = \frac{1}{2} (|(0.25 - \varepsilon) - 0.25| + |0.25 - 0.25| + |(0.25 + \varepsilon) - 0.25| + |0.25 - 0.25|) = \varepsilon.$$

Take a set of statistical tests  $\mathcal{A}_1, \dots, \mathcal{A}_5$  that distinguish the previous probability distributions. Suppose we are given two bits  $\mathbf{b}_0$  and  $\mathbf{b}_1$ . Test  $\mathcal{A}_1$  outputs  $\mathbf{b}_1$ . By the previous information, it is clear that  $\Delta_{\mathcal{A}_1}[X, Y] = |(0.25 + \varepsilon) - 0.25| + |0.25 - 0.25| = \varepsilon$ . Test  $\mathcal{A}_2$  outputs  $\mathbf{b}_0$ , so then  $\Delta_{\mathcal{A}_2}[X, Y] = |0.25 - 0.25| + |0.25 - 0.25| = 0$ . If Test  $\mathcal{A}_3$  outputs  $\mathbf{b}_0 + \mathbf{b}_1 \bmod 2$ , also denoted by the exclusive-or operator  $\mathbf{b}_0 \oplus \mathbf{b}_1$ , its statistical distance is given by  $\Delta_{\mathcal{A}_3}[X, Y] = |0.25 - 0.25| + |(0.25 + \varepsilon) - 0.25| = \varepsilon$ . Test  $\mathcal{A}_4$  outputs  $\mathbf{b}_0 \vee \mathbf{b}_1$ , so  $\Delta_{\mathcal{A}_4}[X, Y] = |0.25 - 0.25| + |(0.25 + \varepsilon) - 0.25| + |0.25 - 0.25| = \varepsilon$ . And finally, if Test  $\mathcal{A}_5$  outputs  $\mathbf{b}_0 \wedge \mathbf{b}_1$ , its statistical distance is  $\Delta_{\mathcal{A}_5}[X, Y] = |0.25 - 0.25| = 0$ .

Based on this information, we can determine that  $\mathcal{A}_1, \mathcal{A}_3$ , and  $\mathcal{A}_4$  are “good” tests with respect to  $\mathbf{D}_1$  and  $\mathbf{D}_2$  because their respective statistical distances are precisely  $\Delta[X, Y]$ . Likewise, tests  $\mathcal{A}_2$  and  $\mathcal{A}_5$  are considered “bad” because they both have statistical distance 0.

## 2.8 Probabilistic Algorithms

Algorithms may use the additional instruction  $x \stackrel{r}{\leftarrow} \{0, 1\}$  for a random variable  $X$  uniform over  $\mathbf{D} = \{0, 1\}$ . Such algorithms are called *probabilistic algorithms* and we say that they “flip coins”.

For any probabilistic algorithm, the set of possible outputs form the support set of a probability distribution. In particular, if  $a \in \{0, 1\}$  is a possible output for a probabilistic algorithm  $\mathcal{A}$  with input  $x$ , we define

$$\text{Prob}[\mathcal{A}(x) = a] = \frac{\#\{b \in \{0, 1\}^n : \mathcal{A} \text{ flips } b \text{ and outputs } a\}}{2^n},$$

where  $n$  denotes the number of coin flips performed by  $\mathcal{A}$  for a given  $x$ . Depending on the specifications of the algorithm, determining  $n$  can be cumbersome. We may assume without loss of generality however, that a probabilistic algorithm  $\mathcal{A}$  makes the same number coin flips for all inputs of the same length. This restriction does not affect the computational power of our underlying probabilistic algorithm model.

**Example.** Consider the following algorithm. Call it  $\mathcal{A}_1$ .

```

1: Input  $1^n$ 
2: select  $x_0, \dots, x_{n-1} \stackrel{r}{\leftarrow} \{0, 1\}$ 
3: if  $\sum_{i=0}^{n-1} 2^i x_i \geq 2^{n-1}$ 
4:   then output 1
5:   else output 0

```

Since 1 is a possible output,

$$\text{Prob}[\mathcal{A}_1(1^n) = 1] = \frac{\#\{b \in \{0, 1\}^n : \mathcal{A} \text{ flips } b \text{ and outputs } 1\}}{2^n} = \frac{2^{n-1}}{2^n} = \frac{1}{2}.$$

**Example.** Call the following algorithm  $\mathcal{A}_2$ .

```

1: Input  $1^n$ 
2: repeat  $n$  times
3:    $x \stackrel{r}{\leftarrow} \{0, 1\}$ 
4:   if  $x = 1$ , output 1 and halt
5:   output Fail

```

Then we have the following probabilities

$$\begin{aligned} \text{Prob}[\mathcal{A}_2(1^n) = \text{Fail}] &= \frac{1}{2^n} \\ \text{Prob}[\mathcal{A}_2(1^n) = 1] &= 1 - \frac{1}{2^n}. \end{aligned}$$

Let  $A$  be a  $n$ -bit number. We call the left-most bit in the binary expansion of  $A$  the *most significant bit*. To avoid trivialities, we require that the most significant bit of  $A$  be 1.

Below are three probabilistic algorithms that attempt to sample the uniform over  $[0, A)$ . To measure the quality of a sampler, one must compute the statistical distance between the sampler’s output distribution and the uniform distribution over the set  $\{0, 1, 2, \dots, A - 1\}$ .

*Exercise:* Consider the following set of samplers. Investigate the output probability distributions of each to determine which has the most uniform distribution.

**Sampler 1:**

---

```

1:  $n := \lceil \log_2 A \rceil$ 
2: choose:  $x_0, x_1, \dots, x_{n-1} \xleftarrow{r} \{0, 1\}$ 
3:  $y := \sum_{i=0}^{n-1} 2^i x_i$ 
4: output  $y \bmod A$ 

```

**Sampler 2:**

```

1: choose:  $x_0, x_1, \dots, x_{A-1} \xleftarrow{r} \{0, 1\}$ 
2:  $y := \sum_{i=0}^{A-1} x_i$ 
3: output  $y$ 

```

**Sampler 3:**

```

1:  $n := \lceil \log_2 A \rceil$ 
2: repeat
3: choose:  $x_0, x_1, \dots, x_{n-1} \xleftarrow{r} \{0, 1\}$ 
4:  $y := \sum_{i=0}^{n-1} 2^i x_i$ 
5: if  $y < A$  output  $y$  and halt
6: else repeat

```

### 3 Symmetric Cryptosystems

Although we will not discuss symmetric cryptosystems in class, here we provide several interesting examples, many of which are of important historical significance.

In a symmetric cryptosystem, both ends of a communication channel share a common secret key. This key is necessary for both the encryption and decryption of messages.

**Definition 3.0.1.** A *symmetric cryptosystem* is composed of the the following elements:

- A plaintext message space  $\mathcal{M}$
- A ciphertext message space  $\mathcal{C}$
- A key space  $\mathcal{K}$
- An efficient encryption algorithm  $\mathcal{E}: \mathcal{K} \times \mathcal{M} \rightarrow \mathcal{C}$
- An efficient decryption algorithm  $\mathcal{D}: \mathcal{K} \times \mathcal{C} \rightarrow \mathcal{M}$
- An efficient key generation algorithm  $\mathcal{G}: \mathbb{N} \rightarrow \mathcal{K}$

In addition to the above, a symmetric cryptosystem must satisfy the *correctness property*:

- For all  $m \in \mathcal{M}$  and  $k \in \mathcal{K}$ ,  $\mathcal{D}(k, \mathcal{E}(k, m)) = m$ .

#### 3.1 Classical ciphers

##### Substitution Ciphers

One of the most basic symmetric cryptosystems is the substitution cipher. In a substitution cipher, the encryption algorithm replaces each message  $m \in \mathcal{M}$  with a corresponding ciphertext  $c \in \mathcal{C}$ . For a given key, the substitution function is a mapping  $\pi: \mathcal{M} \rightarrow \mathcal{C}$  and the decryption algorithm performs the inverse substitution  $\pi^{-1}: \mathcal{C} \rightarrow \mathcal{M}$ .

##### Example (Affine Cipher).

- Message Spaces:  $\mathcal{M}, \mathcal{C} = \mathbb{Z}_N$



- Key Space:  $(a, b) \in \mathbb{Z}_N \times \mathbb{Z}_N$  with  $\gcd(a, N) = 1$
- Encryption Algorithm:  $\mathcal{E}((a, b), m) = am + b \pmod N$

Substitution ciphers, and in particular affine ciphers have been used for thousands of years. One famous affine cipher, known as the Caesar cipher or the shift cipher was used by the Roman emperor Julius Caesar in about 50 BC. In the Caesar cipher,  $a = 1$  and  $b = 3$ , so after assigning each letter of the alphabet to a number, the cipher shifts each letter to the right by 3 (mod 24). In modern times, such a technique cannot withstand frequency statistical analysis attacks. Due to the small number of literate people at that time however, the method sufficed.

Substitution ciphers in  $\mathbb{Z}_N$  can be viewed as permutations on the set  $\{0, 1, \dots, N - 1\}$ . Perhaps the simplest way to encode the English alphabet is to use  $\mathbb{Z}_{26}$ , where each letter is identified with a unique integer modulo 26. The key space is  $26!$ . Unfortunately, this type of cipher is very vulnerable to frequency statistical analysis attacks.

### Polyalphabetic Ciphers

In a polyalphabetic cipher, a plaintext element is repeated and substituted into different ciphertext elements.

#### Example (Vigenère Cipher).

- Key: “gold”
- Plaintext Message: “proceed”
- Encryption Algorithm: Character-wise addition modulo 26
- Decryption Algorithm: Character-wise subtraction modulo 26

To encode “proceed”,

$$\begin{array}{|c|c|c|c|c|c|c|} \hline p & r & o & c & e & e & d \\ \hline g & o & l & d & g & o & l \\ \hline v & f & z & f & k & s & o \\ \hline \end{array} \longleftrightarrow \begin{array}{|c|c|c|c|c|c|c|} \hline 15 & 17 & 14 & 2 & 4 & 4 & 3 \\ \hline 6 & 14 & 11 & 3 & 6 & 14 & 11 \\ \hline 21 & 5 & 25 & 5 & 10 & 18 & 14 \\ \hline \end{array}$$

Polyalphabetic ciphers provide more security against frequency statistical analysis attacks than the previous monoalphabetic ciphers. The polyalphabetic cipher’s main weakness lies instead in the repetitive use of the same key.

### Vernam Cipher and the One-Time Pad

Gilbert Vernam, an engineer for Bell Labs proposed this cipher in 1918.

- Message Spaces:  $\mathcal{M}, \mathcal{C} = \{0, 1\}^n$
- Key Space:  $\mathcal{K} = \{0, 1\}^n$
- Encryption Algorithm:  $\mathcal{E}(k, m) = k \oplus m$
- Decryption Algorithm:  $\mathcal{D}(k, c) = k \oplus c$

This cipher encodes and decodes each element character by character using a previously determined, randomly generated key. Since the key is never reused or repeated, it became known as the one-time pad. The encryption and decryption algorithms are identical, but the properties of the exclusive-or (XOR) operator  $\oplus$  guarantee that the correctness property is satisfied. This cryptosystem is provably secure in the information-theoretical sense. Its main drawback lies in the fact that the key must be at least the length of the original message.

### Transposition Ciphers

A transposition cipher rearranges the positions of each character according to a permutation  $\pi$ . The decryption algorithm recovers the original message by applying the inverse permutation  $\pi^{-1}$ .

#### Example (Transposition Cipher).

- Key:  $\pi = (2143)$
- Plaintext Message: “code”
- Ciphertext Message: “oced”

In this example, the inverse permutation  $\pi^{-1}$  is the same as the encryption permutation.

## 3.2 The Data Encryption Standard (DES)

The Data Encryption Standard (DES) is an algorithm that takes messages of a fixed length and divides them into blocks. The encryption and decryption operations act on these blocks and return outputs of the same length. The system is deterministic and considered to be a polyalphabetic substitution cipher.

The plaintext and ciphertext message spaces are 64-bit strings  $\mathcal{M}, \mathcal{C} = \{0, 1\}^{64}$  and the key space is a 56-bit string  $\mathcal{K} = \{0, 1\}^{56}$ . To encode a message using DES, first divide the message into the 32-bit left and right sub-blocks  $L_0$  and  $R_0$ . Take an initial permutation  $IP$  to be a fixed permutation, independent of the encryption key  $k$ . Then

1.  $(L_0, R_0) \leftarrow IP(input)$
2. Take an S-box function  $f: \{0, 1\}^{48} \times \{0, 1\}^{32} \longrightarrow \{0, 1\}^{32}$  (we will formally define  $f$  later in the section) and 48-bit string keys  $k_1, k_2, \dots, k_{16}$  derived from the 56-bit key  $k$ . Repeat the following operations 16 times:
  - $L_i = R_{i-1}$
  - $R_i = L_{i-1} \oplus f(k_i, R_{i-1})$
3. Output  $\leftarrow IP^{-1}(R_{16}, L_{16})$

The decryption algorithm follows in a similar fashion, with the key schedule reversed.

### Feistel Cipher

Here we show that the iterative operations above satisfy the correctness properties for the DES cyptosystem. Steps 1-3 are collectively known as a **Feistel cipher**. Let  $X_L$  and  $X_R$  represent the left and right 32-bit substrings of the input. Mathematically, this cipher is based on a **round function**  $F_k: \{0, 1\}^{64} \longrightarrow \{0, 1\}^{64}$  given by

$$F_k(X) = X_R \parallel X_L \oplus f(k, X_R).$$

Recall that the concatenation operation  $\parallel$  has the lowest precedence in operations.

In order to express DES as a Feistel cipher, we first define a transposition function,

$$T(X) = X_R \parallel X_L.$$

It should be clear from the above structure that the encryption operation can be described by

$$IP^{-1} \circ T \circ F_{k_{16}} \circ \dots \circ F_{k_2} \circ F_{k_1} \circ IP(X),$$

where  $\circ$  denotes the usual function composition. Similarly, the decryption operation of DES can be viewed as

$$IP \circ F_{k_{16}} \circ F_{k_{15}} \circ \cdots \circ F_{k_1} \circ T \circ IP^{-1}(X).$$

**Lemma 3.2.1.** *Let  $\mathcal{F}$  be the set of all functions  $F: \{0,1\}^{64} \rightarrow \{0,1\}^{64}$ . It follows that for all  $m > 0$  and  $F_1, F_2, \dots, F_m \in \mathcal{F}$ ,*

$$F_1 \circ F_2 \circ \cdots \circ F_m \circ T \circ F_m \circ \cdots \circ F_2 \circ F_1(X) = T(X).$$

*Proof.* We prove this by inducting on the number of functions  $m$ . When  $m = 1$ , we have one function  $F$  in  $\mathcal{F}$ , so

$$\begin{aligned} F \circ T \circ F(X) &= F \circ T(X_R \parallel X_L \oplus f(k, X_R)) \\ &= F(X_L \oplus f(k, X_R) \parallel X_R) \\ &= X_R \parallel X_L \oplus f(k, X_R) \oplus f(k, X_R) \\ &= X_R \parallel X_L \\ &= T(X). \end{aligned}$$

Assume the conclusion holds true for  $i$  functions in  $\mathcal{F}$ : for any  $F_1, F_2, \dots, F_i \in \mathcal{F}$ , it holds that

$$\underbrace{F_1 \circ F_2 \circ \cdots \circ F_i}_{i \text{ functions}} \circ T \circ \underbrace{F_i \circ \cdots \circ F_2 \circ F_1}_{i \text{ functions}}(X) = T(X).$$

Suppose we have an operation with  $i + 1$  functions:

$$F_1 \circ F_2 \circ \cdots \circ F_i \circ F_{i+1} \circ T \circ F_{i+1} \circ F_i \circ \cdots \circ F_2 \circ F_1(X).$$

Note that we insert the new function next to  $T$  because of the indexing, but we are not inducting on the index; we are inducting on the number of functions in the sequence. Using our inductive hypothesis, our expression reduces to our base case:

$$F_1 \circ \underbrace{F_2 \circ \cdots \circ F_i \circ F_{i+1}}_{i \text{ functions}} \circ T \circ \underbrace{F_{i+1} \circ F_i \circ \cdots \circ F_2}_{i \text{ functions}} \circ F_1(X) = F_1 \circ T \circ F_1(X) = T(X).$$

■

**Theorem 3.2.1.** *The DES cryptosystem satisfies the correctness property.*

*Proof.* We prove this for a two-round DES scheme. Using Lemma 3.2.1, the proof easily generalizes for a larger number of rounds. Take any plaintext message  $X$ . The corresponding ciphertext message is  $IP^{-1} \circ T \circ F_2 \circ F_1 \circ IP(X)$ . Our goal is to show that we can recover  $X$  by applying the same function with the key schedule reversed; that is,  $IP^{-1} \circ T \circ F_1 \circ F_2 \circ IP(X)$  inverts  $IP^{-1} \circ T \circ F_2 \circ F_1 \circ IP(X)$ . Noting that  $IP$  and  $IP^{-1}$  are inverses and  $T$  is its own inverse, we have

$$\begin{aligned} IP^{-1} \circ T \circ F_1 \circ F_2 \circ IP \circ IP^{-1} \circ T \circ F_2 \circ F_1 \circ IP(X) &= IP^{-1} \circ T \circ F_1 \circ F_2 \circ T \circ F_2 \circ F_1 \circ IP(X) \\ &= IP^{-1} \circ T \circ T \circ IP(X) \\ &= IP^{-1} \circ IP(X) \\ &= X \end{aligned}$$

by Lemma 3.2.1. ■

This holds true independent of how  $f$  is implemented within each  $F_i$ .

### S-Box Function

In DES, the **S-box** function  $f$  is used to produce a random, non-linear distribution of plaintext messages over the ciphertext message space. Specifically,  $f: \{0, 1\}^{48} \times \{0, 1\}^{32} \rightarrow \{0, 1\}^{32}$  by computing  $f(k, A)$  according to several predetermined components:

1. Expand  $A$  from 32 to 48 bits according to a fixed table.
2. Compute  $k \oplus A$ .
3. Output 8-bit string  $B_1, B_2, \dots, B_8$ , where each  $B_i$  is a 6-bit block.
4. Determine  $[S_1(B_1), \dots, S_8(B_8)]$ , where  $S_i: \{0, 1\}^6 \rightarrow \{0, 1\}^4$  is a fixed substitution map for each  $i$ .
5. Apply a final fixed permutation.

The key schedule  $k_1, k_2, \dots, k_{16}$  is obtained from the original 56-bit encryption key  $k$ , where  $k$  is padded to a 64-bit key  $k'$  by adding a parity bit after every seventh bit of  $k$ . Each  $k_i$  is the substring of  $k'$  used in the  $i$ th iteration.

### The Security of DES

The security of the DES cryptosystem has long been subject to debate. The main criticism focused on the length of the key, which made it vulnerable to brute force attacks. In order to increase the key size, the algorithm could be run multiple times, each time with a different key. Although the key size was considerably large, many prominent cryptographers maintained that the NSA could break the DES encryption by brute force.

## 3.3 The Advanced Encryption Standard (AES)

The current encryption standard for the National Institute of Standards and Technology (NIST) is the Advanced Encryption Standard (AES), also known as Rijndael. Rijndael, pronounced Ran-dahl, was designed by the two Belgian cryptographers Vincent Rijmen and Joan Daeman and was adopted as a standard in 2001 after an extensive five year competition between fifteen designs.

Rijndael is a symmetric block cipher that uses keys of 128, 192, or 256 bits to encrypt and decrypt 128-bit blocks. Here we focus only on a 128-bit key.

To encrypt or decrypt a message, a 128-bit block of plaintext, or respectively ciphertext is divided into 16 bytes,

$$\text{InputBlock} = \langle m_0, m_1, \dots, m_{15} \rangle.$$

The key is divided in a similar fashion,

$$\text{KeyBlock} = \langle k_0, k_1, \dots, k_{15} \rangle.$$

Rijndael operates on the  $4 \times 4$  matrix representations of the InputBlock and KeyBlock:

$$\text{InputBlock} = \begin{bmatrix} m_0 & m_4 & m_8 & m_{12} \\ m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \end{bmatrix}, \quad \text{KeyBlock} = \begin{bmatrix} k_0 & k_4 & k_8 & k_{12} \\ k_1 & k_5 & k_9 & k_{13} \\ k_2 & k_6 & k_{10} & k_{14} \\ k_3 & k_7 & k_{11} & k_{15} \end{bmatrix}.$$

This algorithm, like DES, operates through a number of rounds. In the simplest case, a 128-bit message block and 128-bit key block require 10 rounds.

A round transformation is denoted by  $\text{Round}(\text{State}, \text{RoundKey})$ , where  $\text{State}$  is the  $4 \times 4$  matrix returned by the previous transformation and  $\text{RoundKey}$  is a matrix derived from the InputKey by

some mapping known as the key schedule. When encrypting, the initial *State* is the InputBlock of plaintext and the final *State* outputs the encrypted message. When decrypting, the first *State* is the InputBlock of ciphertext, and the final round returns the original message. Specifically, for any byte  $B = \{0, 1\}^8$ ,

$$\text{Round}: B^{4 \times 4} \times B^{4 \times 4} \longrightarrow B^{4 \times 4},$$

where  $\text{Round}(\text{State}, \text{RoundKey}) = \text{newState}$ .

With the exception of the final round, four internal transformations compose each round transformation:

$$\begin{aligned} \text{Round}(\text{State}, \text{RoundKey}) \{ \\ & \text{SubBytes}(\text{State}); \\ & \text{ShiftRows}(\text{State}); \\ & \text{MixColumns}(\text{State}); \\ & \text{AddRoundKey}(\text{State}, \text{RoundKey}); \\ \} \end{aligned}$$

The final round, denoted  $\text{FinalRound}(\text{State}, \text{RoundKey})$ , differs from the preceding rounds in that it omits the MixColumns operation.

Each round transformation inverts to decrypt. The inverse is denoted using the usual inverse function notation  $\text{Round}^{-1}$  and  $\text{FinalRound}^{-1}$ .

### The Internal Functions of the Rijndael Cipher

Rijndael's internal functions are defined over a binary extension of a finite field. This extension is generated by the ring of all polynomials modulo  $f(X) = X^8 + X^4 + X^3 + X + 1$  over  $\mathbb{F}_2$ . It is necessary to note that  $f(X)$  is irreducible.

Any element in the field can be viewed as a polynomial over  $\mathbb{F}_2$  of degree less than 8, and all operations are performed modulo  $f(X)$ . Each element  $B = \{0, 1\}^8$  can therefore be written as

$$b_7X^7 + b_6X^6 + b_5X^5 + b_4X^4 + b_3X^3 + b_2X^2 + b_1X + b_0,$$

where  $b_i \in \mathbb{F}_2$  is the  $i$ th bit of  $B$ .

**Example (Addition).** Consider the polynomials  $g(X) = X^5 + X^3 + X + 1$  and  $h(X) = X^3 + X^2 + X$ . To compute  $g(X) + h(X) \bmod (f(X), 2)$ ,

$$\begin{aligned} (X^5 + X^3 + X + 1) + (X^3 + X^2 + X) &= X^5 + 2X^3 + X^2 + 2X + 1 \\ &\equiv X^5 + X^2 + 1 \bmod (f(X), 2). \end{aligned}$$

Because these operations are performed over  $\mathbb{F}_2$ , we can also view addition as exonerating the bit values. Writing  $g(X) = 00101011$  and  $h(X) = 00001110$ , we have

$$00101011 \oplus 00001110 = 00100101$$

**Example (Multiplication).** Take the polynomials  $X^3 + X$  and  $X^6 + X + 1$ . Then

$$(X^3 + X)(X^6 + X + 1) = X^9 + X^7 + X^4 + X^3 + X^2 + X$$

Over  $\mathbb{F}_2$  we see  $Xf(X) = X^9 + X^5 + X^4 + X^2 + X$ , so  $X^9 = Xf(X) + X^5 + X^4 + X^2 + X$ . Substituting this in above, we obtain

$$\begin{aligned} &= Xf(X) + X^7 + X^5 + 2X^4 + X^3 + 2X^2 + 2X \\ &\equiv X^7 + X^5 + X^3 \bmod (f(X), 2). \end{aligned}$$

### The SubBytes(*State*) Function

The first internal transformation of the Rijndael cipher performs a nonlinear substitution on each byte of *State* according to an  $8 \times 8$  lookup table *A*. Let  $s_{ij} \in \mathbb{F}_{2^8}$  be a 1-byte element from *State* for  $0 \leq i, j \leq 3$ . After completing the SubBytes step, the *newState* matrix is

$$\text{newState} = \begin{bmatrix} s'_{00} & s'_{01} & s'_{02} & s'_{03} \\ s'_{10} & s'_{11} & s'_{12} & s'_{13} \\ s'_{20} & s'_{21} & s'_{22} & s'_{23} \\ s'_{30} & s'_{31} & s'_{32} & s'_{33} \end{bmatrix},$$

where

$$s'_{ij} = \begin{cases} A \cdot s_{ij}^{-1} \oplus b, & s_{ij} \neq 0 \\ b, & \text{otherwise} \end{cases}$$

for a fixed constant *b*.

### The ShiftRows(*State*) Function

The ShiftRows operation permutes each row of *State*. If

$$\text{State} = \begin{bmatrix} s_{00} & s_{01} & s_{02} & s_{03} \\ s_{10} & s_{11} & s_{12} & s_{13} \\ s_{20} & s_{21} & s_{22} & s_{23} \\ s_{30} & s_{31} & s_{32} & s_{33} \end{bmatrix}, \quad \text{then} \quad \text{newState} = \begin{bmatrix} s_{00} & s_{01} & s_{02} & s_{03} \\ s_{11} & s_{12} & s_{13} & s_{10} \\ s_{22} & s_{23} & s_{20} & s_{21} \\ s_{33} & s_{30} & s_{31} & s_{32} \end{bmatrix}.$$

### The MixColumns(*State*) Function

The MixColumns operation acts on each column of *State*. Let

$$s = \begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix}$$

be any column. We write this as a polynomial of degree 3 over  $\mathbb{F}_{2^8}[X]$ ,

$$s(X) = s_3X^3 + s_2X^2 + s_1X + s_0.$$

Define the fixed cubic polynomial

$$c(X) = c_3X^3 + c_2X^2 + c_1X + c_0 = 03X^3 + 01X^2 + 01X + 02,$$

where  $03, 02, 01 \in \mathbb{F}_{2^8}$ . The MixColumns transformation multiplies  $s(X)$  by  $c(X)$  in  $\mathbb{F}_{2^8}[X]$  modulo the polynomial  $X^4 + 1$ :

$$d(X) = c(X)s(X) \bmod (X^4 + 1, 2^8).$$

The resulting polynomial  $d(X)$  replaces the column  $s(X)$  in the *newState* matrix.

**Lemma 3.3.1.**  $X^i \equiv X^{i \bmod 4} \bmod X^4 + 1$ .

**Lemma 3.3.2.**  $d(X) = d_3X^3 + d_2X^2 + d_1X + d_0$  where

$$d_i = \sum_{k+j \equiv i \pmod{4}} c_k s_j$$

for  $0 \leq k, j \leq 3$ .

**Example.** The coefficient of  $X^2$  in the product  $c(X)s(X) \bmod (X^4 + 1, 2^8)$  is  $d_2 = c_2s_0 + c_1s_1 + c_0s_2 + c_3s_3$ .

Since we are adding and multiplying in  $\mathbb{F}_{2^8}$ , these results can be represented through matrix multiplication in  $\mathbb{F}_{2^8}$ ,

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} c_0 & c_3 & c_2 & c_1 \\ c_1 & c_0 & c_3 & c_2 \\ c_2 & c_1 & c_0 & c_3 \\ c_3 & c_2 & c_1 & c_0 \end{bmatrix} \begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix}.$$

### The AddRoundKey(*State*) Function

In this final transformation, we add the elements of *State* to those of *RoundKey* byte by byte in  $\mathbb{F}_{2^8}$ .

### Decryption

Because each of the four internal functions is invertible, a message is decrypted by applying the inverse encryption operations in reverse order.

```

Round-1(State, RoundKey) {
    AddRoundKey-1(State, RoundKey);
    MixColumns-1(State);
    ShiftRows-1(State);
    SubBytes-1(State);
}

```

The SubBytes function is the crucial step, since it provides the necessary nonlinear element for the cipher. The ShiftRows and MixColumns operations are then used to spread the entropy of the input. While AES does take several steps to complete an encryption, its overall simplicity adds to its appeal.

It is to be noted that Rijndael should use different codes and hardware circuits for encryption and decryption.

## 4 Modes of Operation

Block ciphers process messages of a fixed length by breaking them into fixed-size pieces and operating on each piece. In practice, messages have varying lengths. Different modes of operation allow us to circumvent this issue by adding nondeterminism and padding plaintext to a fixed length invariant. Here we discuss four modes. The following notation will be helpful:

- P: Plaintext Message
- C: Ciphertext Message
- E: Encryption Algorithm
- D: Decryption Algorithm
- IV: Initial Vector

### Electronic Codebook Mode

The electronic codebook mode (ECB) is the simplest mode of operation. A plaintext message is divided into blocks of an appropriate length and each is encrypted individually.

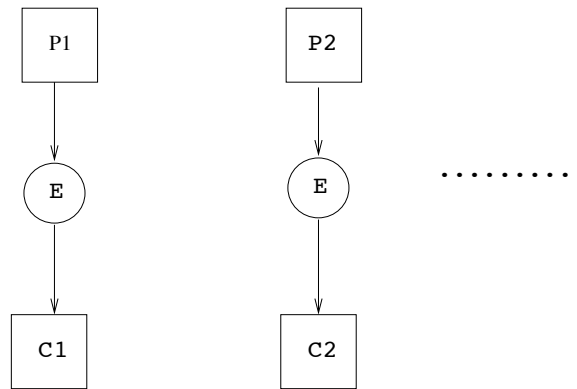


Figure 2: Electronic Codebook Mode

### The Cipher Block Chaining Mode

The cipher block chaining mode (CBC) outputs a sequence of cipher blocks, each dependent on all preceding blocks. The original plaintext is segmented into blocks  $P_1, P_2, \dots$ . The first block  $P_1$  is exonerated with a random  $n$ -bit initial vector before being encrypted as  $C_1$ .  $C_1$  is then exonerated with  $P_2$ . In general,  $C_i$  is the vector used when encrypting  $P_{i+1}$ . One benefit of CBC is that the initial vector need not be kept secret.

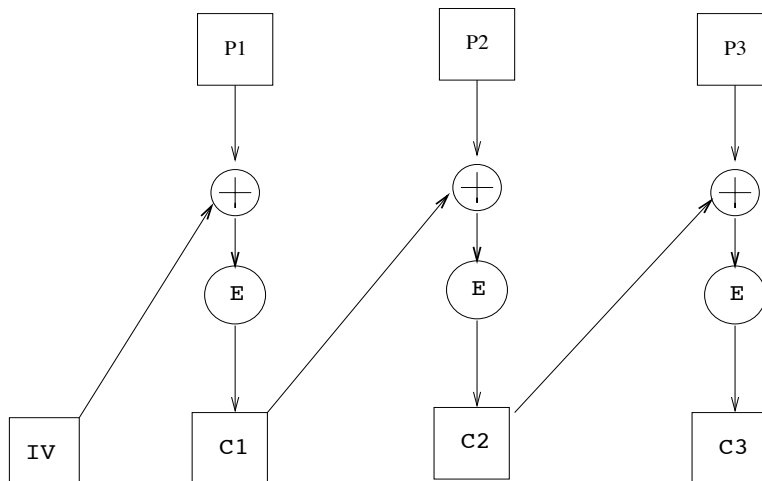


Figure 3: Cipher Block Chaining Mode

### The Counter Mode

The counter mode (CTR) first feeds the encryption algorithm a counter-value. The encrypted counter-value is exonerated with the first block of plaintext  $P_1$  to obtain the cipherblock  $C_1$ . A different counter-value is used for each  $P_i$ , so every cipherblock is independent. Because of this, CTR has the ability to encode all blocks simultaneously or in no particular order.



## The Output Feedback Mode

The output feedback mode (OFB) first encrypts a fixed, random  $n$ -bit initial vector, which it then exonerates with  $P_1$ . When  $P_2$  is added, the encrypted initial vector is again run through the encryption algorithm. In this manner, the encryption algorithm only acts on the initial vector. By repeatedly encrypting  $IV$ , a stream of code is created that interacts with the plaintext. Unlike the counter mode however, the order of the cipherblocks matters.

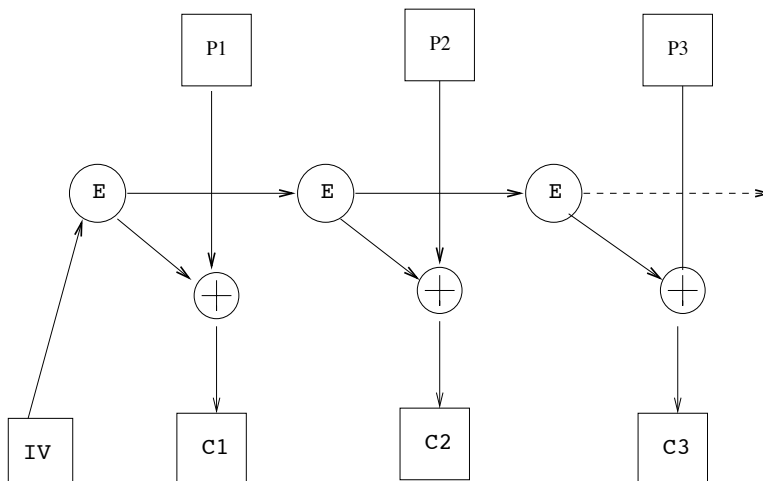


Figure 4: Output Feedback Mode

## 5 Diffie-Hellman Key Exchange Protocol

In 1976, Whitefield Diffie and Martin Hellman published their paper *New Directions in Cryptography*, revolutionizing modern cryptography. Prior to this publication, all significant cryptographic techniques relied on some pre-agreed upon key. In their paper however, Diffie and Hellman proposed a protocol that enabled two parties, having no prior communication, to jointly establish a secret key over an insecure channel. Here we will introduce the concrete key exchange protocol and examine its security in the presence of both passive and active adversaries.

### 5.1 The Diffie-Hellman Protocol

Figure 5 illustrates the concrete Diffie-Hellman key exchange protocol. To begin, two parties, Alice and Bob, choose the values  $x_A$  and  $x_B$  respectively. These can be determined using the coin flipping techniques discussed in Section 2.8. Neither party discloses their value to the other.

The notation  $x \stackrel{\mathcal{I}}{\leftarrow} \mathbb{Z}_m$  means that  $x$  is sampled according to the uniform over  $\mathbb{Z}_m$ . Observe that  $y_B^{x_A} = y_A^{x_B} \pmod p$ , so  $k_A = k_B$  and both parties compute the same value in  $\mathbb{Z}_p^*$ .

In Section 1.1 we mentioned our interest in the goals, designs, primitives, models, and proofs of cryptography. The goal of a key exchange protocol is to establish a key in the presence of an eavesdropper. Our design of interest is the Diffie-Hellman protocol, whose primitives rely on the protocols for sampling random elements. Continuing with this theme, we now naturally want to know how to model the security of the key exchange protocol and investigate the underlying assumptions required for the Diffie-Hellman key exchange to be provably secure.

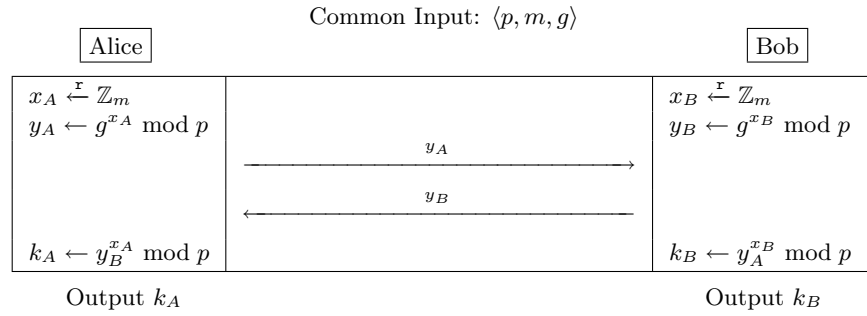


Figure 5: The Diffie-Hellman key exchange protocol, where  $p$  is a large prime and  $g$  is a generator of the group  $\mathbb{Z}_p^*$  of order  $m$ .

## 5.2 Related Number-Theoretical Problems

Here we introduce several potentially hard number theory problems that allow the Diffie-Hellman protocol to reduce. In the following sections, we examine the proper security definition and reduce the security of the protocol to an appropriate number-theoretical assumption.

**Definition 5.2.1.** For a suitable cyclic group  $G = \langle g \rangle$ , take  $y \in G$  of order  $m$ . The *discrete logarithm problem* (DL) is to find an integer  $x \in \mathbb{Z}_m$  such that  $g^x = y$ .

We have no proof that this problem is hard. To the best of our knowledge, the number of steps necessary to find a solution is super-polynomial in the size of the group element, assuming the group is chosen appropriately.

**Definition 5.2.2.** Given a cyclic group  $G = \langle g \rangle$  of order  $m$ ,  $g^a$  and  $g^b$  where  $a, b \xleftarrow{\mathcal{r}} \mathbb{Z}_m$ , the *computational Diffie-Hellman problem* (CDH) is to compute  $g^{ab}$ .

An adversary attacking the Diffie-Hellman protocol does not specifically care about DL. His objective is to solve CDH. It is clear however, that if an adversary could solve DL and derive  $x$  from  $g^x$ , he could solve CDH with a single exponentiation. This therefore establishes a reduction between the discrete logarithm problem and the computational Diffie-Hellman problem:  $\text{CDH} \leq \text{DL}$ .

**Lemma 5.2.1.** *The computational Diffie-Hellman problem is no harder than the discrete logarithm problem.*

It is unknown if the converse holds.

**Definition 5.2.3.** The *decisional Diffie-Hellman problem* (DDH) is as follows: given a group  $G = \langle g \rangle$  of order  $m$  and  $g^a, g^b, g^c$ , where  $a, b, c \xleftarrow{\mathcal{r}} \mathbb{Z}_m$ , decide if  $c = ab$  or  $c \xleftarrow{\mathcal{r}} \mathbb{Z}_m$ .

This is a very weak problem since it only asks an adversary to determine whether or not  $c$  is randomly generated. If an adversary could solve CDH, he could solve DDH by computing  $g^{ab}$  and comparing it to  $g^c$ ; thus,  $\text{DDH} \leq \text{CDH}$ .

**Lemma 5.2.2.** *The decisional Diffie-Hellman problem is no harder than the computational Diffie-Hellman problem.*

Moreover, this last problem is no harder than the discrete logarithm problem.

In the sequel we will show that the Diffie Hellman protocol is secure under an assumption that relates to the DDH problem.

So far we have been conveniently vague in our choice of a group; in fact, we have carefully chosen our parameters to ensure that the underlying problems are indeed hard. The next example

demonstrates this by showing that the discrete logarithm problem is solvable in polynomial-time when we choose an inappropriate group.

**Example.** Consider  $\mathbb{Z}_p^*$  for a large prime  $p$ . By a theorem of Euler,  $\mathbb{Z}_p^*$  has order  $p - 1$ . For this example, consider the case where  $p - 1$  factors into small primes  $q_i$ :  $p - 1 = q_1 q_2 \cdots q_s$ . Then there is a subgroup  $G_i$  of order  $q_i$ .<sup>3</sup> Define the group homomorphism  $f_i: \mathbb{Z}_p^* \rightarrow G_i$  by  $x \mapsto x^{p-1/q_i}$  and let  $g_i = g^{p-1/q_i}$  for some fixed generator  $g$  of  $\mathbb{Z}_p^*$ . Note that  $g_i$  has order  $q_i$ .

Take some  $y = g^x \pmod p$ . Raising both sides to the  $p-1/q_i$  power, we have  $y^{p-1/q_i} \equiv (g^{p-1/q_i})^x \equiv g_i^{x \pmod{q_i}} \pmod p$  where  $1 \leq i \leq s$ . Because  $q_i$  is a small prime, we can use **brute force** to solve the discrete logarithm problem; that is, we can perform an exhaustive search to find the set of congruences  $x_i \equiv x \pmod{q_i}$ . We can then compute  $x$  using the Chinese Remainder Theorem.

To avoid this type of attack, we can select  $\mathbb{Z}_p^*$  such that it contains a large subgroup. For example, if  $p = 2q + 1$  and  $q$  is prime, there is a subgroup of size  $q$ , called the quadratic residue of  $\mathbb{Z}_p^*$ .

**Definition 5.2.4.** The *quadratic residue* of  $G$  is the subgroup of all  $y \in G$  such that there is an  $x \in G$  with  $x^2 = y$ .

When  $G = \mathbb{Z}_n^*$ , we write the quadratic residue as  $QR(n)$ . In the particular case  $G = \mathbb{Z}_p^*$  for a prime  $p$ ,  $QR(p) = \langle g^2 \rangle$  for a generator  $g$  of  $G$ .  $QR(p)$  is exactly half the elements of  $G$ . This is the largest proper subgroup of  $\mathbb{Z}_p^*$ .

The mapping  $x \mapsto x^{\frac{p-1}{2}}$  is particularly useful in this context. It is easy to see that the image of the map is  $\{1, -1\}$ .

We prove the following useful result regarding quadratic residues.

**Lemma 5.2.3.** Consider some  $a \in \mathbb{Z}$  and  $p \equiv 3 \pmod 4$ . It holds that  $a^{\frac{p-1}{2}} = 1 \pmod p$  if and only if  $a \in QR(p)$ .

*Proof.* For the forward direction, suppose that  $a^{\frac{p-1}{2}} = 1 \pmod p$ . Let  $y = a^{\frac{p+1}{4}} \pmod p$ . Then we have

$$y^2 = a^{\frac{p+1}{2}} = a^{\frac{p-1}{2}} \cdot a = a \pmod p$$

Given that  $y^2 = a \pmod p$  we obtain  $a \in QR(p)$ .

For the other direction, if  $a \in QR(p)$ , i.e., we have  $y^2 = a \pmod p$  we have that  $a^{\frac{p-1}{2}} = y^{p-1} = 1 \pmod p$ . ■

Observe that the proof of the lemma provides a way to construct the roots of a quadratic residue modulo  $p$ . Indeed, given  $a$  the two roots of  $a$  modulo  $p$  are calculated as  $\pm a^{\frac{p+1}{4}} \pmod p$ .

### 5.3 Group Generators

**Definition 5.3.1.** A *group generator*  $GGen$  is a probabilistic algorithm that produces a description of a finite group  $G$  when given a length  $\lambda$ . At a minimum, the description contains a group element, the group operation, and a group membership test.

**Example.** Take  $\mathbb{Z}_p$  to be our group for some prime  $p$  of length  $\lambda$ .  $GGen$  returns an element  $g$  of order  $m$ , where  $m$  is some function of  $\lambda$  and  $p$ . The group operation is multiplication modulo  $p$ , and if an integer is between 0 and  $p - 1$ , it passes the group membership test.

For example the algorithm  $GGen$  on input  $1^\lambda$  can calculate a random number  $p$  of the form  $3k + 4$  that has  $\lambda$  bits, and then check if  $p$  is a prime number. If not, it chooses another  $p$  otherwise it checks whether  $(p - 1)/2$  is prime, if not it chooses another  $p$ . When the right  $p$  is found, it chooses a number  $a \in \{2, \dots, p - 2\}$  and random and computes  $a^{(p-1)/2} \pmod p$ . If this value is 1 then it chooses another  $a$ . Otherwise it sets  $g = a^2 \pmod p$ . The output of the algorithm  $GGen$  are the values  $(p, g, m = (p - 1)/2)$ .

<sup>3</sup>The existence of such a subgroup is guaranteed by Cauchy's Theorem.

## 5.4 The Decisional Diffie-Hellman Assumption

Informally, DDH assumes that it is difficult to distinguish between tuples of the form  $\langle g, g^a, g^b, g^{ab} \rangle$  and  $\langle g, g^a, g^b, g^c \rangle$ , where  $g$  belongs to a multiplicative group and  $a, b$ , and  $c$  are randomly chosen exponents.

**Definition 5.4.1.** The group generator  $\text{GGen}$  is said to satisfy the *decisional Diffie-Hellman assumption* provided the following probability ensembles  $\{\mathcal{D}_\lambda\}_{\lambda \in \mathbb{N}}$  and  $\{\mathcal{R}_\lambda\}_{\lambda \in \mathbb{N}}$  are computationally indistinguishable:

$$\begin{aligned} \mathcal{D}_\lambda &:= \left\{ \langle G, m, g \rangle \leftarrow \text{GGen}(1^\lambda); a, b \stackrel{\mathcal{R}}{\leftarrow} \mathbb{Z}_m : \langle G, m, g^a, g^b, g^{ab} \rangle \right\} \\ \mathcal{R}_\lambda &:= \left\{ \langle G, m, g \rangle \leftarrow \text{GGen}(1^\lambda); a, b, c \stackrel{\mathcal{R}}{\leftarrow} \mathbb{Z}_m : \langle G, m, g^a, g^b, g^c \rangle \right\} \end{aligned}$$

where  $m = \text{ord}(g)$ .

Equivalently, if  $\mathcal{A}$  is a statistical test bounded by probabilistic polynomial-time (PPT), it holds that

$$\text{Adv}^{\mathcal{A}}(\lambda) = \left| \text{Prob}_{\gamma \leftarrow \mathcal{D}_\lambda} [\mathcal{A}(\gamma) = 1] - \text{Prob}_{\gamma \leftarrow \mathcal{R}_\lambda} [\mathcal{A}(\gamma) = 1] \right|$$

is negligible in  $\lambda$ .  $\text{Adv}^{\mathcal{A}}$  is called the *advantage* of  $\mathcal{A}$ .

## 5.5 Modeling Security against Passive Adversaries

When defining security, it is important to keep in mind the anticipated adversary. In this section, we focus on passive adversaries. A passive adversary eavesdrops on the communication channel and attempts to extract information about the key without interfering. Before we examine the security definitions, we establish some common notation.

Let  $\text{trans}_{A,B}(1^\lambda)$  be the distribution of the transcripts of the interactions between two players  $A$  and  $B$ . In the Diffie-Hellman protocol, the transcript includes the common input and any exchange of information. The common key produced at the end of a transcript  $\tau$  is denoted  $\text{key}(\tau)$ . Finally, a predicate  $V$  is an algorithm whose only outputs are 1 and 0 (**True** and **False**).

### Security Model 1

The most obvious security model for any key exchange defines the protocol to be secure if an adversary cannot obtain any part of the key. More specifically, for all PPT adversaries<sup>4</sup>  $\mathcal{A}$ ,

$$\text{Prob}_{\tau \leftarrow \text{trans}_{A,B}(1^\lambda)} [\mathcal{A}(\tau) = \text{key}(\tau)]$$

is a negligible function in  $\lambda$ . Under this model, it is plausible for an adversary to obtain all but a small amount of information about the key; it is therefore inadequate. The number of bits protected by this model can be as few as  $\log^2(\lambda)$ .

<sup>4</sup>We say adversary to mean any PPT algorithm.

### Security Model 2

For all PPT adversaries  $\mathcal{A}$  and predicates  $V$ , we define a key exchange to be secure if

$$\text{Prob}_{\tau \leftarrow \text{trans}_{\mathcal{A}, \mathcal{B}}(1^\lambda)} [\mathcal{A}(\tau) = V(\text{key}(\tau))] \leq \frac{1}{2} + \text{negl}(\lambda)$$

for some negligible function  $\text{negl}(\lambda)$ . This model is ideal in that, if our protocol is secure, an adversary cannot identify any information about the key space. Unfortunately, this is also unrealistic.

Assume this model does define security and there is a PPT adversary  $\mathcal{A}$  capable of breaking the key exchange protocol. Then there is a predicate  $V$  such that

$$\text{Prob}_{\tau \leftarrow \text{trans}_{\mathcal{A}, \mathcal{B}}(1^\lambda)} [\mathcal{A}(\tau) = V(\text{key}(\tau))] \geq \frac{1}{2} + \alpha,$$

where  $\alpha$  is nonnegligible. Let  $\mathcal{B}$  be a DDH distinguisher such that, given  $\gamma = \langle G, m, g, a, b, c \rangle$ ,  $\mathcal{B}$  uses  $\gamma$  to form a transcript  $\tau_\gamma = \langle G, m, g, a, b \rangle$ .  $\mathcal{B}$  then simulates  $\mathcal{A}$  on  $\tau_\gamma$  to obtain its output  $S$ .  $\mathcal{B}$  will return 1 if  $V(c) = S$  and 0 if  $V(c) \neq S$ . When  $c$  is a random element of the cyclic group  $G$ , let  $\text{Prob}[V(c) = 1] = \delta$ .

1. If  $\gamma \leftarrow \mathcal{D}_\lambda$ , then  $c = \text{key}(\tau_\gamma)$  and  $\text{Prob}_{\gamma \leftarrow \mathcal{D}_\lambda} [\mathcal{B}(\gamma) = 1] \geq \frac{1}{2} + \alpha$ .
2. If  $\gamma \leftarrow \mathcal{R}_\lambda$ , then  $c \stackrel{\mathcal{R}}{\perp} G$  and

$$\begin{aligned} & \text{Prob}_{\gamma \leftarrow \mathcal{R}_\lambda} [\mathcal{B}(\gamma) = 1] \\ &= \text{Prob}_{\langle G, m, g, a, b, c \rangle \leftarrow \mathcal{R}_\lambda} [\mathcal{A}(G, m, g, a, b) = V(c)] \\ &= \text{Prob}[\mathcal{A}(\tau_\gamma) = V(c)] \\ &= \text{Prob}[\mathcal{A}(\tau_\gamma) = V(c) \mid V(c) = 1] \cdot \text{Prob}[V(c) = 1] + \dots \\ & \quad \dots + \text{Prob}[\mathcal{A}(\tau_\gamma) = V(c) \mid V(c) = 0] \cdot \text{Prob}[V(c) = 0] \\ &= \text{Prob}[\mathcal{A}(\tau_\gamma) = 1] \cdot \text{Prob}[V(c) = 1] + \text{Prob}[\mathcal{A}(\tau_\gamma) = 0] \cdot \text{Prob}[V(c) = 0] \end{aligned}$$

In the special case where  $\delta = 1/2$ , we see

$$\text{Prob}_{\gamma \leftarrow \mathcal{R}_\lambda} [\mathcal{B}(\gamma) = 1] = (\text{Prob}[\mathcal{A}(\tau_\gamma) = 1] + \text{Prob}[\mathcal{A}(\tau_\gamma) = 0]) \frac{1}{2} = \frac{1}{2}.$$

Looking at the DDH assumption,

$$\text{Adv}^{\mathcal{B}} \geq \left( \frac{1}{2} + \alpha \right) - \frac{1}{2} = \alpha.$$

Because  $\alpha$  is nonnegligible,  $\mathcal{B}$  can break the DDH assumption when it has  $\mathcal{A}$  and  $\delta = 1/2$ . When  $\delta \neq 1/2$  however, it is easy to find a  $V$  that the adversary can guess with probability better than  $1/2$  (e.g.,  $V$  can be the “or” of the first two bits of  $c$ ). As a result, all schemes fail under this unreasonably strong model.

### Security Model 3

Finally, we explore a model under which the security of the key exchange protocol can be proven. This will define passive security.

We have to acknowledge that an adversary can distinguish some part of the key, so let

$$\text{Prob}_{key \leftarrow \text{Key}(1^\lambda)} [V(key) = 1] = \delta,$$

where  $\text{Key}(1^\lambda)$  is the key space probability distribution for the protocol with parameter  $1^\lambda$  (i.e., the random variable  $key(\text{trans}_{A,B}(1^\lambda))$ ). It is not hard to see that for the case of Diffie Hellman key exchange it holds that  $\text{Key}(1^\lambda)$  equals a uniformly distributed element of the group  $\langle g \rangle$ . We now define the key exchange protocol is secure provided that

$$\text{Prob}_{\tau \leftarrow \text{trans}_{A,B}(1^\lambda)} [\mathcal{A}(\tau) = V(key(\tau))] \leq \max\{\delta, 1 - \delta\} + \text{negl}(\lambda).$$

Assume

$$\text{Prob}_{\gamma \leftarrow \mathcal{D}_\lambda} [\mathcal{B}(\gamma) = 1] \geq \max\{\delta, 1 - \delta\} + \alpha$$

for nonnegligible  $\alpha$ . Using this, we can show  $\text{Prob}_{\gamma \leftarrow \mathcal{R}_\lambda} [\mathcal{B}(\gamma) = 1] \leq \max\{\delta, 1 - \delta\}$ :

$$\begin{aligned} & \text{Prob}_{\gamma \leftarrow \mathcal{R}_\lambda} [\mathcal{B}(\gamma) = 1] \\ &= \text{Prob}[A(\tau_\gamma) = 1] \cdot \text{Prob}[V(c) = 1] + \text{Prob}[A(\tau_\gamma) = 0] \cdot \text{Prob}[V(c) = 0] \\ &= \text{Prob}[A(\tau_\gamma) = 1]\delta + \text{Prob}[A(\tau_\gamma) = 0](1 - \delta) \\ &\leq \text{Prob}[A(\tau_\gamma) = 1](\max\{\delta, 1 - \delta\}) + \text{Prob}[A(\tau_\gamma) = 0](\max\{\delta, 1 - \delta\}) \\ &= (\text{Prob}[A(\tau_\gamma) = 1] + \text{Prob}[A(\tau_\gamma) = 0]) \max\{\delta, 1 - \delta\} \\ &= \max\{\delta, 1 - \delta\}. \end{aligned}$$

Based on the above, we have proved the following theorem.

**Theorem 5.5.1.** *If the DDH assumption is true, the Diffie-Hellman key exchange protocol is secure against passive adversaries under Security Model 3.*

## 5.6 Suitable Group Generators for the DDH Assumption

In this section, we examine the DDH assumption over two groups.

First consider  $\langle g \rangle = \mathbb{Z}_p^*$  for a large prime  $p$ . This group is potentially a poor choice; in fact, we can construct a PPT algorithm  $\mathcal{A}$  as in Figure ?? that breaks the DDH assumption.

By Euler's Theorem,  $\mathbb{Z}_p^*$  has order  $m = p - 1$ . Since  $p$  is odd for all primes greater than 2,  $m$  is even for any nontrivial group.

Let  $\gamma = \langle p, m, g, a, b, c \rangle$  where  $a = g^x$ ,  $b = g^y$ , and  $c = g^{xy}$ . If  $x$  is even, write  $x = 2k$  for some  $k \in \mathbb{Z}$ . Then

$$a^{m/2} = (g^x)^{m/2} = g^{km} = 1. \text{ If } x \text{ is odd, write } x = 2j + 1 \text{ for some } j \in \mathbb{Z}. \text{ Then } a^{m/2} = (g^{2j+1})^{m/2} = g^{m/2} = -1.$$

Algorithm  $\mathcal{A}(p, m, g, a, b, c)$   
 if  $(a^{m/2} = 1 \vee b^{m/2} = 1) \wedge (c^{m/2} = 1)$   
 then output 1  
 else output 0

Figure 6: A PPT algorithm that breaks the DDH assumption when  $\langle g \rangle = \mathbb{Z}_p^*$ ,  $a, b, c \in \langle g \rangle$ , and  $m = \text{ord}(g)$  is even.

The same result holds for  $g^y$  depending on if  $y$  is even or odd. The parity of  $xy$  clearly depends on the parity of  $x$  and  $y$ , so  $c^{m/2} = (g^{xy})^{m/2} = 1$  as long as one of  $x$  or  $y$  is even. Thus,

$$\text{Prob}_{\gamma \leftarrow \mathcal{D}}[\mathcal{A}(\gamma) = 1] = \frac{3}{4}.$$

If instead  $\gamma \leftarrow \mathcal{R}$ , so  $c = g^z$  for a randomly chosen  $z$ , there is an equal probability that  $z$  will be even or odd. So

$$\text{Prob}_{\gamma \leftarrow \mathcal{R}}[\mathcal{A}(\gamma) = 1] = \frac{3}{8}.$$

Based on this information,

$$\text{Adv}^{\mathcal{A}} = \frac{3}{4} - \frac{3}{8} = \frac{3}{8}.$$

In an ideal situation, both probabilities are close to  $1/2$ , so their difference is negligible. Since  $\text{Adv}^{\mathcal{A}} = 3/8$ ,  $\mathcal{A}$  can distinguish between the two tuples. It is therefore ineffective to build a key exchange over  $\mathbb{Z}_p^*$ .

One group we can build a key exchange over is the quadratic residue  $QR(p)$  of  $\mathbb{Z}_p^*$ . For example, if  $p = 2q + 1$  for a prime  $q$ ,  $QR(p)$  has order  $q$ . To the best of our knowledge, this is an adequate group. Recall that  $QR(p) = \langle g^2 \rangle$  for a generator  $g$  of  $\mathbb{Z}_p^*$ , so  $QR(p)$  is a cyclic group of odd order.

## 5.7 Modified Diffie-Hellman Protocol

Under the DDH assumption, the generated key is a random element from a group  $\langle g \rangle$  whose structure we know very little about (to see the problem consider writing an algorithm that has the objective to produce 10 uniformly distributed bits with only source of randomness coming from the operation  $y \stackrel{r}{\leftarrow} \langle g \rangle$ ). This becomes problematic when using the key in cryptographic applications. Here we look at how to extract a random integer from a random group element. This is useful in that we do understand the structure of integers.

One approach is to define a predicate  $V$  such that  $\text{Prob}_{x \leftarrow \langle g \rangle}[V(x) = 1] = 1/2$ .  $V$  then defines one unpredictable bit from the adversary's point of view. It is unclear however, how to find even one such predicate. One must completely understand the structure of the group in order to discern a random bit. Instead, take  $p = 2m + 1$  and define the map

$$H: \mathbb{Z}_m \longrightarrow QR(p)$$

by  $x \mapsto (x + 1)^2 \bmod p$ . This is a bijection. To show it is injective, assume  $H(x) = H(y)$  for some  $x, y \in \mathbb{Z}_m$ . Then

$$(x+1)^2 \equiv (y+1)^2 \pmod{p}$$

$$(x+1)^2 - (y + 1)^2 \equiv 0 \pmod p$$

$$x^2 + 2x - 2y - y^2 \equiv 0 \pmod p$$

$$(x-y)(x+y+2) \equiv 0 \pmod p.$$

So either  $x - y \equiv 0 \pmod p$  or  $x + y + 2 \equiv 0 \pmod p$ . Since  $x, y \in \mathbb{Z}_m$ , we have  $0 \leq x, y \leq m - 1$ . Then

$$x+y+2 \leq 2(m - 1) + 2 = 2m$$

$$;2m+1 \equiv 0 \pmod p.$$

Thus  $x + y + 2 \not\equiv 0 \pmod p$ , which leaves only  $x - y \equiv 0 \pmod p$ , or equivalently  $x \equiv y \pmod p$ . Since  $x, y \in \mathbb{Z}_m \subset \mathbb{Z}_p$ , it holds that  $x = y$ , showing  $H$  is injective.  $H$  is surjective by the following pre-image of any  $y \in QR(p)$ ,

$$H^{-1}(y) = \begin{cases} y^{p+1/4} \pmod{p-1}, & \text{if } y^{p+1/4} \pmod{p} \in \{1, 2, \dots, m\} \\ p - y^{p+1/4} \pmod{p-1}, & \text{otherwise.} \end{cases}$$

Using this, we can modify the key exchange protocol as is seen in Figure ??.

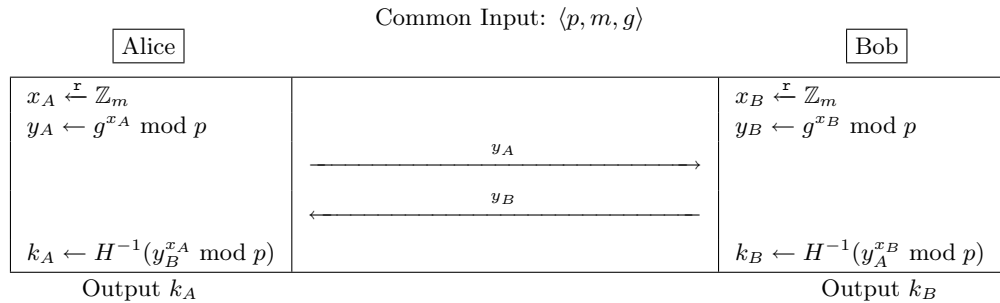


Figure 7: The modified Diffie-Hellman key exchange protocol where  $p$  is a large prime,  $g$  generates the group  $QR(p)$  of order  $m$ , and  $H: \mathbb{Z}_m \rightarrow QR(p)$  by  $x \mapsto (x + 1)^2 \pmod p$ .

Under the modified Diffie-Hellman key exchange protocol, we can now use the bijection  $H$  to pass from a random element from a group whose structure we do not fully understand to a random integer modulo  $m$ .

*Exercise:* We have shown how to derive a random element from  $\mathbb{Z}_m$ . This enables us to access cryptographic applications requiring a random integer modulo  $m$  as a key. Most applications however, necessitate that the key be a bit string. Determine how to extract the longest possible bit string from an integer modulo  $m$ .



It is interesting to note that in a  $\lambda$ -bit key, the probability that the least significant bit is 1 is very close to  $1/2$ , while the probability that the most significant bit is 1 can be far from  $1/2$ .

### 5.8 Stronger Adversaries

While the Diffie-Hellman key exchange protocol, as given in Section ??, is secure against an eavesdropper, it does not remain so against a more active adversary. In Figure 3, we show the **man-in-the-middle attack** in which the adversary, Malorie, participates in the exchange of information between Alice and Bob. The adversary is now the communication channel itself. Malorie can inject messages into the conversation and impersonate the identity of each party to the other. In doing so, Malorie creates two keys, one to share with Alice and one to share with Bob.

This attack exemplifies the need to authenticate and verify authentication on each exchange. Next we introduce a digital signature, which is an important cryptographic primitive, essential in defending against tactics like the man-in-the-middle attack.

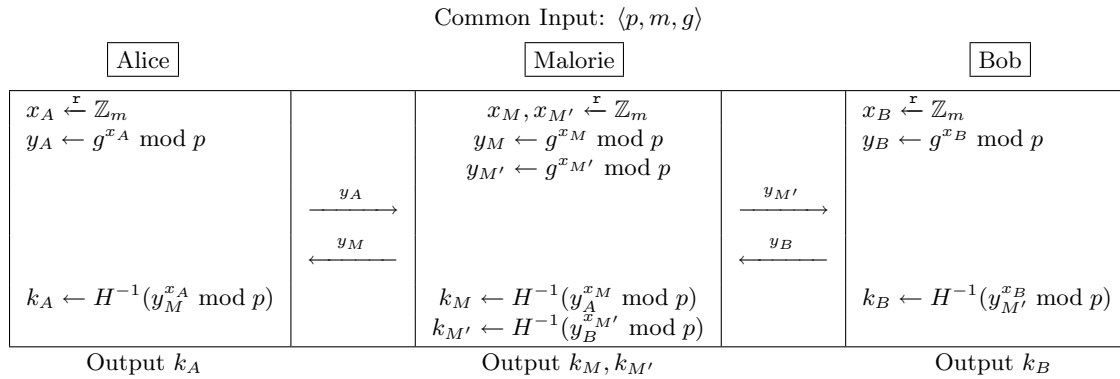


Figure 8: The “man-in-the-middle” attack on the Diffie-Hellman key exchange protocol.

## 6 Digital Signatures

A digital signature is a fundamental cryptographic primitive, technologically equivalent to a hand-written signature. In many applications, digital signatures are used as building blocks in larger cryptographic protocols and systems.

In a signature scheme, each party holds a unique signing key  $sk$  that uniquely signs a message  $M$ . Each party publishes their corresponding public verification key  $pk$ . Only someone with knowledge of  $sk$  can sign a message, but all parties have access to  $pk$  and can verify a signature. Such schemes are useful in that they prevent someone with just a verification key from computing a signing key with more than a negligible probability. Moreover, it is unfeasible for an adversary to produce a valid message-signature pair associated to a verification key.

**Definition 6.0.1.** A *digital signature scheme* is a triple of algorithms  $(\text{Gen}, \text{Sign}, \text{Verify})^5$  such that

- The key generation algorithm **Gen**: Take a security parameter  $1^\lambda$  and output the pair  $(pk, sk)$ . We call  $pk$  the public or verification key and  $sk$  the secret or signing key.

<sup>5</sup>Gen and Sign are PPT algorithms, Verify is a deterministic polynomial-time algorithm.

- The signing algorithm **Sign**: When given a security parameter  $1^\lambda$ , a signing key  $sk$ , and a message  $M$ , produce the digital signature  $\sigma$  of  $M$ .
- The verification algorithm **Verify**: Take a verification key  $vk$ , a digital signature  $\sigma$  and a message  $m$ . Return **True**=1 or **False**=0 to indicate if the signature is valid.

The primary goal of digital signatures is **unforgeability**, or rather that a PPT adversary cannot create a new valid message-signature pair. The strongest attack against digital signatures is called the **chosen method attack**. In such an attack, an adversary has unlimited access to a signing oracle that returns signatures to messages of the adversary's choosing.

**Definition 6.0.2.** A digital signature scheme  $(\text{Gen}, \text{Sign}, \text{Verify})$  is said to have **unforgeability against chosen message attacks** (UF-CMA) if for every PPT adversary  $\mathcal{A}$  that accesses a signing oracle  $\text{Sign}(sk, \cdot)$   $\ell$  times, the probability that  $\mathcal{A}$  produces  $\ell + 1$  distinct valid message-signature pairs is negligible. When the messages are distinct, we say there is **strong unforgeability**. When the message-signature pairs are distinct, we say there is **regular unforgeability**.

## 6.1 The RSA Function: The $e$ th Power Map on $\mathbb{Z}_n^*$

The RSA cryptosystem was developed in 1977 at MIT by Ron Rivest, Adi Shamir, and Leonard Adleman. It was the first public-key encryption scheme that could both encrypt and sign messages. As with the Diffie-Hellman key exchange protocol, the system enabled two parties to communicate over a public channel.

Suppose Alice decides to send our dear friend Bob a message. In order to facilitate a private conversation over an insecure channel, Bob selects and publishes the integers  $n$  and  $e$ . Alice writes her message  $x$  and computes

$$E(x) = x^e \bmod n,$$

known as the  **$e$ th power map** of  $x$ . She then sends  $y = E(x)$  to Bob, who in order to see the message, must compute the  $e$ th root of  $y$ . This is believed to be hard, as we discussed in Section 5.2. If Bob selects  $n$  and  $e$  appropriately however, there is an alternate method. We will see that Bob can apply the  $d$ th power map to  $y$  to recover  $x$ ,

$$D(y) = y^d = x^{ed} \equiv x^{1+\varphi(n)k} \equiv x \bmod n$$

where  $k \in \mathbb{Z}_n$  and the Euler function  $\varphi(n)$  is defined as follows:

**Definition 6.1.1.** For  $n \in \mathbb{N}$ , the **Euler function**  $\varphi(n)$  counts the number of integers in  $\mathbb{Z}_n$  relatively prime to  $n$ :

$$\varphi(n) = \# \{k \in \mathbb{Z}_n : \gcd(k, n) = 1\}.$$

Equivalently,  $\varphi(n)$  is the number of invertible elements in  $\mathbb{Z}_n$ :

$$\varphi(n) = \# \{k \in \mathbb{Z}_n : k\ell = 1 \text{ for some } \ell \in \mathbb{Z}_n\}.$$

To compute Euler's function,

$$\varphi(n) = \begin{cases} p^e - p^{e-1}, & n = p^e \text{ for prime } p \\ \prod_{i=1}^j \varphi(p_i^{e_i}), & n = p_1^{e_1} \cdots p_j^{e_j} \text{ for distinct primes } p_i. \end{cases}$$

When  $n = p^e$ , it is easy to count the number of integers modulo  $n$  that  $p$  does not divide. Extending  $\varphi$  to a composite integer  $n = p_1^{e_1} \cdots p_j^{e_j}$  falls from the fact that  $\varphi$  is multiplicative on relatively prime integers:  $\varphi(mn) = \varphi(m)\varphi(n)$  when  $\gcd(m, n) = 1$ . This can be shown by proving

$$\mathbb{Z}_{mn}^* \cong \mathbb{Z}_m^* \times \mathbb{Z}_n^*$$

using the Chinese Remainder Theorem.

We are interested in the special case where  $n$  is the product of two large primes  $p$  and  $q$ . The multiplicative group  $\mathbb{Z}_n^* = \{a \in \mathbb{Z}_n : \gcd(a, n) = 1\}$  consists of  $\varphi(n) = (p-1)(q-1)$  elements. In order for the above protocol to be effective (i.e. for Bob and no one else to know  $d$ ), take  $e$  to be a prime such that  $1 \leq e \leq \varphi(n)$  and  $\gcd(e, \varphi(n)) = 1$ . Then the  $e$ th power map

$$E: \mathbb{Z}_n^* \longrightarrow \mathbb{Z}_n^*,$$

defined by  $x \mapsto x^e \bmod n$  is invertible. In particular, when  $ed \equiv 1 \pmod{\varphi(n)}$ , the  $d$ th power map  $D$  inverts  $E$ . If Bob chooses  $e$  and  $\varphi(n)$  carefully, he can easily recover  $d$  using the Euclidean algorithm.

It is clear that the strength of this technique lies primarily in Bob's choice of  $n$ . If  $p$  and  $q$  are obvious, any interested party can compute  $\varphi(n)$  and therefore  $d$ . Likewise, given  $n$  and  $\varphi(n)$ , one can compute  $d$ . This implies that finding  $e$ th roots in  $\mathbb{Z}_n^*$  relies on the factorization of  $n$ . Since the factorization problem is believed to be hard, the RSA function appears to be difficult to invert in polynomial-time. This then constitutes a one-way function. The **RSA assumption** states that it is hard to invert the RSA function.

## 6.2 RSA Digital Signatures

After its introduction, the security of the RSA signature scheme was argued in the Full-Domain Hash setting and was proven to be secure in the Random Oracle Model based on the RSA assumption.

In general, a **hash function** is a mapping that takes a message of arbitrary length and returns an element of bounded size.

The ideal hash function should be easily computable, noninvertible, and behave like an injection in the sense that it is extremely unlikely for two messages, no matter how similar to map to the same string or hash. In this section, we will assume our hash function  $H$  to be ideal with the range

$$H: \{0, 1\}^* \longrightarrow \mathbb{Z}_n^*,$$

where  $\{0, 1\}^* = \bigcup_{k=0}^{\infty} \{0, 1\}^k$ .

In the RSA signature scheme,

- The key generation algorithm **Gen**: First choose two random primes  $p$  and  $q$  such that  $|p| = |q| = \lambda$ . Compute  $n = pq$  and  $\varphi(n) = (p-1)(q-1)$ . Second, choose a random prime  $e < \varphi(n)$  such that  $\gcd(e, \varphi(n)) = 1$  and compute  $d \equiv e^{-1} \pmod{\varphi(n)}$ . The verification key is  $(n, e)$  and the signing key is  $d$ . A full-domain hash function  $H$  is available to all parties.
- The signing key **Sign**: Given  $d$  and a message  $M$ , output the digital signature  $\sigma = H(M)^d \bmod n$ .
- The verification algorithm **Verify**: Given  $(n, e)$  and  $(M, \sigma)$ , verify that  $\sigma^e = H(M) \bmod n$ . If equality holds, the result is **True**; otherwise the result is **False**.

We call it a collision when two messages have the same signature under a hash function. These instances produce forgeries. For example, if  $(M, \sigma)$  is a valid message-signature pair and  $m'$  is a message such that  $H(M) = H(M')$ , then  $(M', \sigma)$  is also a valid message-signature pair. In order for the RSA assumption to hold,  $H$  must satisfy some form of collision resistance. This is proved in the so called Random Oracle Model.

A **random oracle** is a function that produces a random looking output for each query it receives. It must be consistent with its replies: if a question is repeated, the random oracle must return the same answer. The absence of any concrete structure in random oracles makes them useful in cryptographic applications when abstracting a hash function. If a scheme is secure assuming the adversary views some function as a random oracle, it is said to be secure in the **Random Oracle Model**.

Figure ?? illustrates how a hash function  $H$  is modeled as a random oracle.

- Given  $M \notin \text{History}$ , choose  $t \xleftarrow{x} \mathbb{Z}_n^*$  and enter  $(M, t)$  in  $\text{History}$ . Return  $t$ .
- Given  $M$  such that  $(M, t) \in \text{History}$  for some  $t$ , return  $t$ .

Figure 9: A random oracle, where  $\text{History}$  represents the set of all  $(\text{input}, \text{output})$  pairs previously served by the oracle.

We will prove that the RSA signature scheme is secure in the Random Oracle Model under the RSA assumption.

**Theorem 6.2.1.** *Under the Random Oracle Model, any PPT forgery adversary that attacks the RSA digital signature scheme and finds a forged signature with probability  $\alpha$  can be transformed into an algorithm that finds  $e$ th roots with probability at least*

$$\frac{1}{q_H} \cdot \left( \alpha - \frac{1}{2^\lambda} \right),$$

where  $\lambda$  is the security parameter and  $q_H$  is the number of queries made to the random oracle  $H$ .

*Proof.* Let  $\mathcal{A}$  be an adversary that finds a forged signature with probability  $\alpha$ . We will create an algorithm  $\mathcal{B}$  that solves the RSA problem.

Say  $\mathcal{B}$  is given input  $\langle n, e, y \rangle$ , where  $n$  and  $e$  are obtained through the RSA key generation algorithm  $\text{Gen}$  and  $y$  is chosen at random from  $\mathbb{Z}_n^*$ .  $\mathcal{B}$ 's goal is to find  $z \in \mathbb{Z}_n^*$  such that  $z = y^{1/e} \pmod n$ . To do this,  $\mathcal{B}$  forms the verification key  $(n, e)$  and gives it to  $\mathcal{A}$ .  $\mathcal{A}$  will make both signing and random oracle queries, both of which  $\mathcal{B}$  must answer. We assume  $\mathcal{B}$  knows  $q_H$ . This is not a problem as  $\mathcal{A}$  is polynomial-time bounded and  $q_H$  is smaller than the number of steps in the execution of  $\mathcal{A}$ . Figure ?? illustrates how  $\mathcal{B}$  operates when given access to  $\mathcal{A}$ .

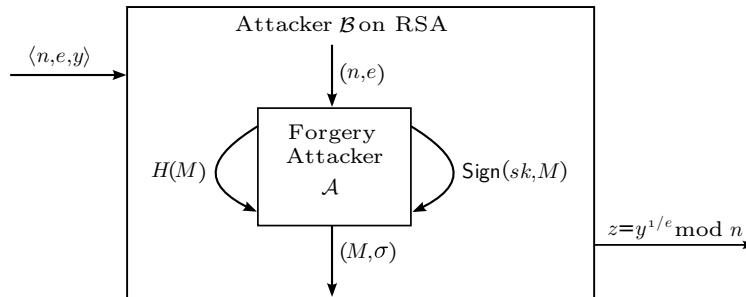


Figure 10: The attacker  $\mathcal{B}$  must simulate  $H$  and  $\text{Sign}$  to use attacker  $\mathcal{A}$ .

First suppose  $\mathcal{A}$  does not invoke the signing algorithm  $\text{Sign}$ , so  $\mathcal{A}$  produces  $(M, \sigma)$  after making  $q_H$  queries to the random oracle  $H$ .  $\mathcal{B}$  answers these queries by simulating  $H$  as in Figure ?. By our assumption,  $\sigma$  is a valid forged signature for  $M$  with probability  $\alpha$ . Assume it is valid, so  $\sigma = H(M)^d \pmod n$ . If  $H(M) = y$ , then  $\sigma = H(M)^d = y^d \pmod n$  and  $\sigma$  is an  $e$ th root of  $y$  (since  $d \equiv e^{-1} \pmod{\varphi(n)}$ ).

Observe that the above is highly unlikely, as the chances  $H(M) = y$  are minuscule. This is because there is no relation between the random oracle simulation by  $\mathcal{B}$  and the value  $y$ . In order to obtain this result with some reasonable likelihood,  $\mathcal{B}$  deviates from Figure ?? when answering the random oracle queries.  $\mathcal{B}$  chooses a random number  $j$  in  $\{1, \dots, q_H\}$  and attempts to answer the  $j$ th random oracle query with  $y$ :  $H(M) = y$ . In particular, the random oracle is modified to be parameterized by  $j, y$  and operates as follows:

Maintain a query counter.

- Given  $M \notin \text{History}$ : if this is the  $j$ th query, set  $t = y$ , else choose  $t \xleftarrow{r} \mathbb{Z}_n^*$ . Enter  $(M, t)$  in *History*. Return  $t$ .
- Given  $M$  such that  $(M, t) \in \text{History}$  for some  $t$ , return  $t$ .

Figure 11: A modified random oracle simulation as used by algorithm  $\mathcal{B}$  to “plug-in” a challenge  $y$  into the oracle’s responses.

In what follows we denote the output forged message of the adversary by  $M$ , and the queries to the signing oracle by  $M_1, \dots, M_{q_S}$ . As discussed above, we initially assume  $q_S = 0$ , i.e., there are no queries to the signing oracle.

Consider the event  $E$ , that  $m \in \text{History}$ , and the event  $\neg E$ , the complement. Also let  $S$  be the event that the adversary  $\mathcal{A}$  successfully produces a forged signature (i.e., the verification algorithm is executed on  $M, \sigma$  and outputs 1). Based on the theorem statement we know that  $\text{Prob}[S] = \alpha$ . On the other hand, observe that  $\text{Prob}[S | \neg E] \leq 1/\varphi(n) \leq 2^{-\lambda}$ . This is the case since in the conditional space  $\neg E$ , the adversary has not asked  $M$  to  $H$  and thus the value of  $H(M)$  is undetermined until the verification algorithm is invoked. Since the adversary has already produced  $\sigma$ , the chances that  $H(M)^e = \sigma$  in  $\mathbb{Z}_n^*$  are  $1/\varphi(n) \leq 2^{-\lambda}$ .

Now in the conditional space on  $E$ , there is a  $1/q_H$  chance the random oracle will correctly guess the query on which  $M$  is asked. We call this the event  $G$ . If  $G$  occurs, it follows that  $H(M) = y$ , i.e.,  $y$  would be plugged into the right location. In such an occasion, the algorithm  $\mathcal{B}$  would have successfully recovered the  $e$ th root of  $y$ . We call this the event  $V$ . We next provide a lower bound on  $V$ . First we have that  $\text{Prob}[S | \neg E] \leq 2^{-\lambda}$ , therefore

$$\text{Prob}[S \wedge \neg E] = \text{Prob}[S | \neg E] \cdot \text{Prob}[\neg E] \leq 2^{-\lambda}.$$

Based on this we obtain the lower bound

$$\text{Prob}[S \wedge E] \geq \alpha - 2^{-\lambda}.$$

Next we split the probability space according to  $E$  and calculate the probability of  $V$  as follows:

$$\text{Prob}[V] = \text{Prob}[V|E] \cdot \text{Prob}[E] + \text{Prob}[V \wedge \neg E] \geq \text{Prob}[S \wedge G|E] \cdot \text{Prob}[E].$$

Due to the independence of the events  $S, G$  in the conditional space  $E$ , we have that

$$\text{Prob}[V] \geq \text{Prob}[S|E] \cdot \text{Prob}[G|E] \cdot \text{Prob}[E] = \text{Prob}[S \wedge E] \cdot \text{Prob}[G|E] = \frac{\alpha - 2^{-\lambda}}{q_H}.$$

This completes the argument in the case  $q_S = 0$ . We next consider the general case where  $q_S$  is a polynomial in  $\lambda$ . In addition to making random oracle queries,  $\mathcal{A}$  now asks  $\mathcal{B}$  to sign a message  $M'_i$ .  $\mathcal{B}$  must answer in a way that is consistent with the random oracle queries: if  $\mathcal{B}$  returns  $\sigma_i$ , it holds that  $\sigma_i = H(M'_i)^d \bmod n$ , so  $\sigma_i^e = H(M'_i) \bmod n$ . This implies  $(M'_i, \sigma_i^e)$  is in *History*. We can accommodate this by again modifying the simulation of  $H$  performed by  $\mathcal{B}$  as seen in Figure ??.

Now when asked to sign  $M_i$ ,  $\mathcal{B}$  can first ask its random oracle subroutine for  $M_i$  and then consult the *History* table it maintains for a record of the form  $(M_i, t_i, \rho_i)$ . Unless  $\rho_i = \diamond$  it can then proceed

Maintain a query counter.

- Given  $M \notin \text{History}$ : if this is the  $j$ th query, set  $t = y$ ,  $\rho = \diamond$ ,  
     else choose  $\rho \xleftarrow{\$} \mathbb{Z}_n^*$  and set  $t = \rho^e \bmod n$ .  
     Enter  $(M, t, \rho)$  in *History*. Return  $t$ .
- Given  $M$  such that  $(M, t, \rho) \in \text{History}$  for some  $t$ , return  $t$ .

Figure 12: A second modified random oracle simulation as used by algorithm  $\mathcal{B}$  to “plug-in” a challenge  $y$  into the oracle’s responses while keeping the “pre-images” of the oracles responses under the  $e$ th power map.

to answer the signing query with  $\rho_i$ . Observe that the signing oracle simulation is perfect as long as  $\rho_i \neq \diamond$ , since  $\rho_i^e = t_i \bmod n$ , i.e.,  $\rho_i$  is an  $e$ th root of  $t_i = H(M_i)$  in  $\mathbb{Z}_n^*$ . Here we are not overly concerned with the case  $\rho_i = \diamond$  as it means that the guess of  $\mathcal{B}$  for  $M$  is mistaken (due to the condition that a successful forgery must be on a message that  $\mathcal{A}$  does not query to the signing oracle) and thus the simulation is already doomed.

To finish the proof we need to ensure that the adversary  $\mathcal{A}$  cannot detect any difference in the random oracle responses as defined in Figure ???. Indeed observe that now the values returned to the adversary are of the form  $\rho^e \bmod n$ , as opposed to random values  $t$  selected uniformly from  $\mathbb{Z}_n^*$ . This nevertheless turns out to not be an issue since the RSA function is a bijection on  $\mathbb{Z}_n^*$  and as a result,  $\rho^e \bmod n$  is a random variable that is uniformly distributed over  $\mathbb{Z}_n^*$  under the assumption that  $\rho$  is uniformly distributed. ■

## 7 Zero-Knowledge Proofs

A **proof of knowledge** is a protocol that enables one party to convince another of the validity of a statement. In a **zero-knowledge proof**, this is accomplished without revealing any information beyond the legitimacy of the proof. We will examine several examples of zero-knowledge proofs and then formalize a definition. We begin our discussion by looking at the general formulation.

We have two parties, the prover  $\mathcal{P}$  and the verifier  $\mathcal{V}$ .  $\mathcal{P}$  must convince  $\mathcal{V}$  that she has some knowledge of a statement  $x$  without explicitly stating what she knows. We call this knowledge a **witness**  $w$ . Both parties are aware of a predicate  $R$  that will attest to  $w$  being a valid witness to  $x$ . In general,

- The predicate  $R$  is assumed to be polynomial-time computable: given a witness  $w$  for a statement  $x$ , one can efficiently test that  $R(x, w) = 1$ .
- The prover  $\mathcal{P}$  has  $R, x$ , and  $w$  such that  $R(x, w) = 1$ . She wishes to prove possession of  $w$  by producing a proof of knowledge  $\pi$ .
- The verifier  $\mathcal{V}$  has  $R, x$ , and  $\pi$ .

In order for the above protocol to be useful in cryptographic applications, we can make the following assumptions.

- Given  $R$ , it is hard to find a corresponding  $w$  such that  $R(x, w) = 1$ .
- The prover  $\mathcal{P}$  is reluctant to reveal  $w$ ; otherwise the solution is trivial.
- The verifier  $\mathcal{V}$  can efficiently check the validity of  $\pi$ .

## 7.1 Examples of Zero-Knowledge Proofs

To demonstrate these concepts, we present two simple examples.

**Example (Where’s Waldo).** In the game *Where’s Waldo*, there is a large board depicting an intricate scene of characters, all of whom resemble “Waldo”. The objective of the game is to discern Waldo from amongst the look-alikes.

Suppose the old comrades, Alice and Bob decide to play. Alice claims to have found Waldo’s exact location, but she does not want to show Bob. After all, antagonizing one’s opponent makes many games more entertaining.

Here the assumption that Waldo exists is the statement, Waldo’s  $(x, y)$  coordinates are the witness, and the procedure of receiving  $(x, y)$  and verifying that Waldo is indeed there relates to the predicate  $R$ .

One possible solution, and admittedly not the only one, is for Alice to cover the board with a large piece of paper with a small, Waldo-sized hole in its center. To prove she has found Waldo, Alice moves the paper so that Waldo, and nothing else, is visible through the hole. Note that for this solution to be effective, the dimensions of the paper must be at least twice those of the board.

**Example (Magic Door).** After being soundly beaten in *Where’s Waldo*, Bob convinces Alice to go spelunking. The two eventually come to a cave as depicted in Figure

???. At the bottom of the cave, there is a magic door that can only be opened using a secret password. Bob proposes a new game to prove to Alice he can open the magic door.

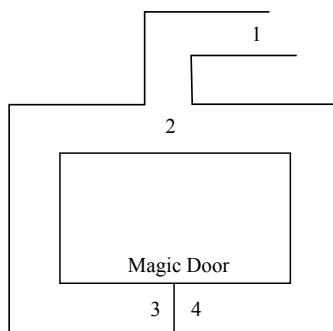


Figure 13: The door between Points 3 and 4 can only be opened using a secret password.

1. Alice stands at Point 1.
2. Bob enters the cave and stands either at Point 3 or 4.
3. After Bob disappears, Alice walks to Point 2.
4. Alice calls to Bob, asking him to come out either the left or the right passage.
5. Bob complies, using the secret password if necessary.
6. Alice and Bob repeat Steps 1-5  $k$  times.

This game illustrates how a probabilistic procedure can be used to model protocols of interest. In particular, after  $k$  repetitions, Bob can convince Alice with probability  $1 - 1/2^k$  that he knows the magic words.

We now present two practical examples that use proofs of knowledge.

**Example.** We return to the  $NP$  class: the set of all problems for which a candidate solution can be verified in polynomial-time. We call a set of strings a **language**. Let  $x$  denote any problem statement and let  $R$  represent a polynomial-time predicate. Then a language  $L$  is in  $NP$  if

$$L = \{x : R(x, w) = 1 \text{ for some } w\}.$$

Take the language  $CLIQUE = \{\langle G, k \rangle : G \text{ is a graph with a clique of size } k\}$ . The witnesses are the sets of  $k$  vertices forming a clique, and the polynomial-time predicate  $R$  verifies that the vertices form a clique.

Another language is  $SAT = \{\langle \Phi \rangle : \Phi \text{ is a satisfiable boolean formula}\}$ . One can check in polynomial-time that a set of variables assigned to  $\Phi \in SAT$  satisfies  $\Phi$ . Zero-knowledge proofs can be used to prove a specific element is in  $CLIQUE$  or  $SAT$ . We will address how in Section ??

**Example.** One key application for zero-knowledge proofs is in user identification schemes. In traditional password mechanisms, an eavesdropping adversary can obtain enough information to gain unauthorized access in a system. In order to allay this problem, suppose the system contains a public directory that assigns a statement of a theorem to each user. In order to gain access to the system, a user must produce a proof of their theorem. Assuming that only an authorized user knows a witness to their proof, a zero-knowledge proof can convince the system of the proofs authenticity. This is directly related to the Schnorr Protocol, which we will discuss in section ??.

## 7.2 Three Basic Properties

Formalizing the definition of a proof of knowledge is a very delicate task. The following definition emerged after fifteen years of work, and has since been deemed an intellectual achievement.

**Definition 7.2.1.** Let  $\langle \mathcal{P}, \mathcal{V} \rangle$  be a pair of interactive programs. Define  $\text{out}_{\mathcal{P}, \mathcal{V}}^{\mathcal{P}}(x, w, z)$  to be the output of  $\mathcal{P}$  when both  $\mathcal{P}$  and  $\mathcal{V}$  are executed with the public input  $x$  and private inputs  $w$  and  $z$  ( $\mathcal{P}$  determines  $w$  and  $\mathcal{V}$  chooses  $z$ );  $\text{out}_{\mathcal{P}, \mathcal{V}}^{\mathcal{V}}$  is similarly defined for  $\mathcal{V}$ . The PPT interactive protocol  $\langle \mathcal{P}, \mathcal{V} \rangle$  is a **zero-knowledge proof** for a language  $L \in NP$  with knowledge error  $\kappa$  and zero-knowledge distance  $\varepsilon$  if the following properties hold.

- **Completeness:** If  $x \in L$  and  $R(x, w) = 1$  for some witness  $w$ , then  $\text{out}_{\mathcal{P}, \mathcal{V}}^{\mathcal{V}}(x, w, z) = 1$  for all strings  $z$  with overwhelming probability  $\nu$ .
- **Soundness:** For any polynomial-time program  $\mathcal{P}^*$

$$\pi_{x, w, z} = \text{Prob}[\text{out}_{\mathcal{P}^*, \mathcal{V}}^{\mathcal{V}}(x, w, z) = 1].$$

A protocol  $\langle \mathcal{P}, \mathcal{V} \rangle$  satisfies soundness if for all  $\mathcal{P}^*$  there exists a probabilistic Turing machine (PTM) program  $K$ , called a **knowledge extractor** with the following property. Suppose that

$$\tilde{\pi}_{x, w, z} = \text{Prob}[K(x, w, z) = w' : R(x, w') = 1].$$

Then it holds that  $\pi_{x, w, z}$  is nonnegligible implies that  $\tilde{\pi}_{x, w, z}$  is nonnegligible.

- **(Statistical) Zero-Knowledge (SZK):** For each polynomial-time program  $\mathcal{V}^*$ , there is a PTM program  $S$ , called the **simulator**, such that for all  $x, w$  with  $R(x, w) = 1$ , the random variables  $S(x, z)$  and  $\text{out}_{\mathcal{P}, \mathcal{V}^*}^{\mathcal{V}^*}(x, w, z)$  are statistically indistinguishable for all strings  $z$ :

$$\forall \mathcal{A} \left| \text{Prob}[\mathcal{A}(S(x, z)) = 1] - \text{Prob}[\mathcal{A}(\text{out}_{\mathcal{P}, \mathcal{V}^*}^{\mathcal{V}^*}(x, w, z)) = 1] \right| < \varepsilon.$$



Completeness is very similar to correctness. Assuming both the prover and verifier follow the protocol faithfully, completeness guarantees that the protocol will succeed with a sufficiently high probability.

The intention of soundness ensures that the protocol will fail when executed by a prover using a false witness and an honest verifier. This is a minimal requirement. The formal definition above asserts something much stronger. It guarantees that a knowledge extractor  $K$  can derive a valid witness from any convincing prover. This implies  $K$  should have some more power than the verifier. In particular,  $K$  has access to the program of the prover, something that the verifier does not (the verifier is a program that interacts with the prover, whereas the knowledge extractor is a program that is derived from the program of the prover).

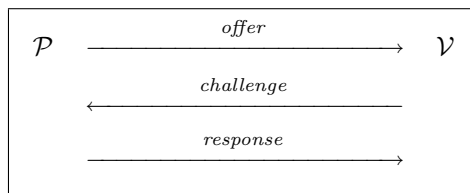
We note that our formulation of soundness is a bit more restrictive (albeit much simpler) than previous formulations in the literature, as it will fail protocols that allow a substantial cheating probability for the prover (e.g.,  $1/2$ ). In most interesting cases such protocols can be made to satisfy our definition through parallel or sequential repetition.

Intuitively, statistical zero-knowledge is a property that prohibits a verifier from extracting information from an honest prover. If the verifier is able to learn anything, there must be an algorithm that simulates the protocol without access to a witness. Moreover, the execution of the algorithm is indistinguishable from that of the protocol.

A weaker version of zero-knowledge is *honest-verifier zero-knowledge* (HVZK). Here it is assumed that the verifier executes the protocol faithfully, but makes additional computations. Specifically, this is captured in the definition above by restricting  $V^*$  to simulate the verifier  $V$  and in the end, simply output the whole communication transcript. Achieving this much weaker property is sometimes also referred to as *semi-honest* verifier zero-knowledge. Even though this relaxes the SZK specifications, it can be used to obtain zero-knowledge proofs in situations employing generic methods. Proving honest verifier zero-knowledge boils down to producing accepting protocol transcripts that are indistinguishable from the honest prover-verifier transcripts, without the use of a witness.

### 7.3 The Schnorr Protocol

One classic three-move protocol that exhibits the properties of a zero-knowledge proof is the Schnorr protocol, also known as the  $\Sigma$ -protocol.



The Schnorr protocol operates over a cyclic group  $G = \langle g \rangle$  of order  $m$ . From our previous discussion,  $\mathcal{P}$  and  $\mathcal{V}$  have group generators  $\langle p, m, g \rangle$ . The prover  $\mathcal{P}$  chooses a witness  $w \in \mathbb{Z}_m$  such that  $h = g^w \pmod p$  for some  $h \in \langle g \rangle$ . The verifier  $\mathcal{V}$  is given  $p, m, g$  and  $h$ , and must confirm that  $w = \log_g h$ .

This can also be described as a language. Define  $\text{DLOG} = \{ \langle p, m, g \rangle, h \} : h = g^w \pmod p \text{ for some } w \in \mathbb{Z}_m \}$ . (DLOG stands for “discrete logarithm”.) Under the Schnorr protocol, there is a very efficient way to prove any statement  $\langle p, m, g \rangle, h$  is in DLOG without revealing  $w = \log_g h$ .

1.  $\mathcal{P}$  chooses  $t \xleftarrow{\mathcal{R}} \mathbb{Z}_m$  and sends  $y = g^t$  to  $\mathcal{V}$ .
2.  $\mathcal{V}$  chooses a challenge  $c \xleftarrow{\mathcal{R}} \mathbb{Z}_m$  and sends  $c$  to  $\mathcal{P}$ .
3.  $\mathcal{P}$  computes  $s = t + wc \pmod m$  and sends  $s$  to  $\mathcal{V}$ .  $\mathcal{V}$  checks and accepts if and only if  $g^s = yh^c$ .

If both the prover and the verifier are honest, it holds that

$$g^s = g^{t+wc} = g^t(g^w)^c = yh^c.$$

The Schnorr protocol is therefore complete and can always convince an honest verifier.

This protocol instigates a special case of the soundness property. Before we formalize an extraction algorithm, let us look at how we can obtain information from a convincing prover  $\mathcal{P}$ .

Suppose we are capable of generating two accepting conversations from  $\mathcal{P}$  with the challenge values  $c \neq c'$ :  $\langle y, c, s \rangle$  and  $\langle y, c', s' \rangle$ . If both  $s$  and  $s'$  are valid, then  $g^s = yh^c$  and  $g^{s'} = yh^{c'}$ . By solving both equations for  $y$  we obtain

$$\begin{aligned} y &= g^s h^{-c} = g^{s'} h^{-c'} \\ h^{c-c'} &= g^{s-s'} \\ h &= g^{(s-s')/(c-c')} \end{aligned}$$

While this does not justify how we can reverse-engineer  $\mathcal{P}$  to obtain the second conversation, it does show that we can extract the witness as  $(s-s')/(c-c') \bmod m$ . We will assume we have access to  $\mathcal{P}$  in a way that allows us to stop at any given point, return to a previous step, and re-simulate the operation.

It is helpful to view the probabilistic program  $\mathcal{P}$  in two steps:

1.  $\mathcal{P}(\text{first}, \langle p, m, g \rangle, h)$  outputs  $\langle y, \text{aux} \rangle$
2.  $\mathcal{P}(\text{second}, \langle p, m, g \rangle, c, \text{aux})$  outputs  $\langle s \rangle$

where  $\text{aux}$  represents the internal information  $\mathcal{P}$  uses, but does not publish. Using this, we can develop a knowledge extractor  $K$  with the following structure:

1. Let  $\rho_1 \stackrel{\text{r}}{\leftarrow} \{0, 1\}^{\lambda_1}$  be the coin tosses required by the first step of  $\mathcal{P}$ . Fix the randomness of  $\mathcal{P}$  with  $\rho_1$  and simulate  $\mathcal{P}(\text{first}, \langle p, m, g \rangle, h)$  to obtain  $y$ .
2. Choose  $c \stackrel{\text{r}}{\leftarrow} \mathbb{Z}_m$ .
3. Let  $\rho_2 \stackrel{\text{r}}{\leftarrow} \{0, 1\}^{\lambda_2}$  be the coin tosses required by the second step of  $\mathcal{P}$ . Simulate  $\mathcal{P}(\text{second}, \langle p, m, g \rangle, c, \text{aux})$  with fixed randomness  $\rho_2$  to obtain  $s$ .
4. Choose  $c' \stackrel{\text{r}}{\leftarrow} \mathbb{Z}_m$  and  $\rho'_2 \stackrel{\text{r}}{\leftarrow} \{0, 1\}^{\lambda_2}$ . Repeat steps 2 and 3 to obtain  $s'$ ; output  $\langle y, c, s \rangle$  and  $\langle y, c', s' \rangle$ .

If the knowledge extractor obtains two accepting conversations, we can directly reconstruct the witness as previously discussed. It remains to show that the knowledge extractor produces two accepting conversations with an adequate probability. To prove this, we need the following lemma.

**Lemma 7.3.1 (Splitting Lemma).** *Let  $X$  and  $Y$  be finite sets. Call  $A \subseteq X \times Y$  the set of **good elements** of  $X \times Y$ . Suppose there is a lower bound on the number of good elements such that*

$$|A| \geq \alpha |X \times Y|.$$

*Define the set of **super-good elements**  $A'$  to be the subset of  $A$  such that*

$$A' = \left\{ (x, y) \in A : k_x > \frac{\alpha}{2} |Y| \right\}$$

*where  $k_x$  is the number of  $y \in Y$  such that  $(x, y) \in A$  for a fixed  $x$ . Then*

$$|A'| \geq \frac{\alpha}{2} |X \times Y|.$$

*Proof.* We prove this by contradiction. Assume  $|A'|/|X \times Y| < \alpha/2$ , so

$$|A| = |A'| + |A \setminus A'| < \frac{\alpha}{2}|X \times Y| + |A \setminus A'|. \quad (2)$$

For any  $(x, y) \in A \setminus A'$ , we have that  $k_x \leq (\alpha/2)|Y|$ . Since there are only  $|X|$  distinct  $x$ s,  $|A \setminus A'| \leq (\alpha/2)|X||Y|$ . From (??) we now obtain

$$|A| < \frac{\alpha}{2}|X \times Y| + \frac{\alpha}{2}|X||Y|.$$

This contradicts the lower bound on  $|A|$ , so  $|A'| \geq \alpha/2|X \times Y|$ . ■

Returning now to the efficiency of our knowledge extractor  $K$ , define

$$X \times Y = \left\{ (\rho_1, (c, \rho_2)) : \rho_1 \in \{0, 1\}^{\lambda_1}, (c, \rho_2) \in \mathbb{Z}_m \times \{0, 1\}^{\lambda_2} \right\}.$$

If the prover is successful with at least probability  $\alpha$ , we define  $A$  to be the set of  $(\rho_1, (c, \rho_2))$  that the verifier accepts. Then  $|A| \geq \alpha|X \times Y|$ . This suggests we can fix a good sequence  $(\rho_1, (c, \rho_2))$  in  $A$  so that the resulting conversation from  $K$  is accepting. By Lemma ??,  $K$  hits a super-good sequence in steps 1 through 3 with probability  $\alpha/2$ .

Suppose the knowledge extractor does hit a super-good sequence. Then there is again an  $\alpha/2$  probability that  $K$  hits another super-good sequence when repeating in Step 4. The probability that both conversations are accepting is therefore  $\alpha^2/4$ . Moreover, there is only a  $1/m$  chance that  $K$  will generate the same challenge values  $c = c'$ .

Consider now the following: let  $S$  be the event that the knowledge extractor is successful. Next let  $C$  be the event that  $c \neq c'$  in the second choice, let  $D$  be the event that the sequence  $(\rho_1, (c, \rho_2))$  is super-good, and let  $E$  be the event that the sequence  $(\rho_1, (c', \rho'_2))$  is good.

It follows that

$$\text{Prob}[S] \geq \text{Prob}[C \wedge D \wedge E] \geq \text{Prob}[D \wedge E] - \text{Prob}[-C] = \frac{\alpha^2}{4} - \frac{1}{m}.$$

This proves that the Schnorr protocol satisfies the soundness property.

Our three-move protocol satisfies honest-verifier zero-knowledge. To show this, we present an algorithm capable of simulating an accepting conversation between an honest prover and a (semi) honest verifier. Suppose that when given the public predicate information  $\langle\langle p, m, g \rangle, h \rangle$ , the simulator  $S$  randomly chooses  $c$  and  $s$  from  $\mathbb{Z}_m$  and outputs  $\langle g^s h^{-c}, c, s \rangle$ . Recall that the output in the honest model is  $\langle g^t, c, t + wc \bmod m \rangle$  where  $t, c \stackrel{\mathcal{R}}{\leftarrow} \mathbb{Z}_m$ . One can easily verify that the two probability distributions are identical; thus, HVZK holds.

**How to go beyond HVZK.** While HVZK is a relatively weak property, it is useful in extending protocols to SZK. There are two generic ways to do this employing two kinds of commitment schemes, both of which rely on the apparent importance of the prover choosing  $y$  independent from  $c$ .

In the first method, the verifier is the first to send a message. Before receiving  $y$ ,  $\mathcal{V}$  chooses  $c$  and computes  $(c', \sigma) \leftarrow \text{Commit}(c)$ .  $\mathcal{V}$  then sends  $c'$  to the prover. When  $\mathcal{P}$  sends  $y$ , the verifier returns  $(c, \sigma)$  to open the commitment. If  $\text{Verify}(c, c', \sigma) = 0$ , the prover stops the protocol; otherwise the protocol is completed using the challenge  $c$ .

The commitment scheme satisfies the binding property, so  $\mathcal{V}$  cannot change  $c$ . Statistical zero-knowledge therefore holds. The hiding property is satisfied by the commitment scheme, so  $\mathcal{P}$  cannot obtain any information about  $c$  so soundness is not violated. A simulator for the zero-knowledge property must then be able to extract  $c$  from  $c'$ . This is called the *extractable* property for a commitment.

In the second method, the prover is once again the first to send a message. After computing  $y$ ,  $\mathcal{P}$  computes  $(y', \sigma) \leftarrow \text{Commit}(y)$  and sends  $y'$  to  $\mathcal{V}$ . Once  $\mathcal{V}$  returns  $c$ ,  $\mathcal{P}$  sends  $(y, \sigma, s)$  to open the commitment. If  $\text{Verify}(y, y', \sigma) = 0$ , the verifier stops the protocol.

Since the commitment scheme satisfies the hiding property,  $y'$  contains no useful information about  $y$ .  $\mathcal{V}$  is therefore forced to pick  $c$  independent from  $y$  as desired. This scheme satisfies the zero-knowledge property.

Note that soundness is not violated under this scheme because the binding property prohibits  $\mathcal{P}$  from opening  $y$  in two different ways. In this setting then, a simulator must be able to bypass the binding property on the commitment; that is, if the simulator commits an arbitrary  $y^*$ , after receiving the challenge  $c$ , it can choose  $y = g^s h^{-c}$ . Commitments that allow this type of violation are called *equivocal*.

Observe that the first scheme added a new move to the protocol, whereas the second maintained the standard three-move structure. For this reason, the second scheme is sometimes preferred. Still the above discussion merely reduces the problem to the design of commitment schemes with either the equivocal or the extractable property for the simulator.

## 7.4 Non-Interactive Zero-Knowledge Proofs

We now introduce a non-interactive version of the Schnorr identification scheme based on what is known as the *Fiat-Shamir Heuristic*.

To make a proof non-interactive, we use a hash function  $H: \{0, 1\}^* \rightarrow \mathbb{Z}_m$  such that a conversation  $\langle y, c, s \rangle = \langle g^t, H(g^t), t + H(g^t)w \bmod m \rangle$ . This mandates that  $c$  be selected after  $y$ , implicitly relying on the properties of the hash function.

To show that SZK holds, assume  $H$  is a random oracle controlled by the simulator. In the Random Oracle Model, a dishonest verifier  $\mathcal{V}^*$  may make queries to the random oracle. Figure ?? illustrates how  $\mathcal{V}^*$  interacts with  $H$ .

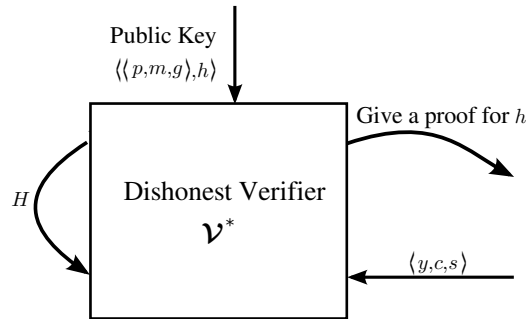


Figure 14: The simulation of a dishonest verifier  $\mathcal{V}^*$  in the Random Oracle Model.

When the verifier asks for a proof of  $h = g^w$ , the simulator randomly chooses  $c$  and  $s$  to calculate  $y = g^s h^{-c}$ . It enters  $(y, c)$  in *History* and returns  $\langle y, c, s \rangle$ . The dishonest verifier cannot distinguish between an honest prover and the simulator unless  $(y, c') \in \text{History}$  with  $c \neq c'$ . Then  $\mathcal{V}^*$  succeeds with probability  $(1/m)q_H$ , where  $q_H$  is the number of random oracle queries.

Next consider soundness in the Random Oracle Model. As with the Schnorr protocol, we want to generate two accepting conversations with the same  $y$  and different challenge values. Using these two conversations, we can extract a witness. Note that  $c = H(y)$ . If a dishonest prover  $\mathcal{P}^*$  makes a single query to the random oracle before producing  $\langle y, c, s \rangle$ , the analysis is same as with the interactive protocol. Problems arise however when  $\mathcal{P}^*$  makes more than one query.

Suppose in the initial round  $\mathcal{P}^*$  makes  $q_H$  queries before outputting a conversation. When a knowledge extractor returns  $\mathcal{P}^*$  to a previous step, there is no guarantee that  $\mathcal{P}^*$  will again

make  $q_H$  queries. When  $\mathcal{P}^*$  terminates, it could return  $\langle y', c', s' \rangle$  with  $c' = H(y')$  and  $y \neq y'$ . This reduces our ability to extract a witness, so we must adjust the probability of obtaining two accepting conversations with the same  $y$ .

Assume that after making  $q_H$  queries,  $\mathcal{P}^*$  chooses one query and uses the corresponding response from the random oracle in its output. Let  $\text{Prob}[A] = \alpha$  denote the probability that the resulting conversation is accepting. Let  $\text{Prob}[Q_i] = \beta_i$  represent the probability that the dishonest prover completes the  $i$ th conversation with  $1 \leq i \leq q_H$ . Define  $\text{Prob}[A \cap Q_i] = \alpha_i$ , then

$$\sum_{i=1}^{q_H} \alpha_i = \alpha \quad \text{and} \quad \sum_{i=1}^{q_H} \beta_i = 1.$$

Define  $\text{Prob}[E]$  to be the probability of extracting a witness from  $\mathcal{P}^*$ . We then have that

$$\text{Prob}[A \cap Q_i] = \text{Prob}[A \mid Q_i] \cdot \text{Prob}[Q_i]$$

so  $\text{Prob}[A \mid Q_i] = \alpha_i / \beta_i$ . From our calculations in Section ??, we obtain

$$\text{Prob}[E \mid Q_i] \geq \frac{\text{Prob}[A \mid Q_i]^2}{4} - \frac{1}{m} = \frac{\alpha_i^2}{4\beta_i^2} - \frac{1}{m}.$$

The overall probability is calculated as follows.

$$\begin{aligned} \text{Prob}[E] &= \sum_{i=1}^{q_H} \text{Prob}[E \mid Q_i] \cdot \text{Prob}[Q_i] \\ &\geq \sum_{i=1}^{q_H} \left( \frac{\alpha_i^2}{4\beta_i^2} - \frac{1}{m} \right) \beta_i \\ &= \frac{1}{4} \sum_{i=1}^{q_H} \frac{\alpha_i^2}{\beta_i} - \sum_{i=1}^{q_H} \frac{\beta_i}{m} \\ &= \frac{1}{4} \sum_{i=1}^{q_H} \frac{\alpha_i^2}{\beta_i} - \frac{1}{m} \sum_{i=1}^{q_H} \beta_i \\ &= \frac{1}{4} \sum_{i=1}^{q_H} \frac{\alpha_i^2}{\beta_i} - \frac{1}{m} \\ &\geq \frac{1}{4q_H} \left( \sum_{i=1}^{q_H} \alpha_i \right)^2 - \frac{1}{m} \\ &= \frac{\alpha^2}{4q_H} - \frac{1}{m} \end{aligned}$$

We can therefore conclude that given a convincing prover, we can extract a witness with probability at least

$$\frac{\alpha^2}{4q_H} - \frac{1}{m}.$$

## 7.5 Honest-Verifier Zero-Knowledge for all NP

Anything in  $NP$  can be proven in a three-move honest-verifier zero-knowledge protocol. Consider the language HC of all Hamiltonian graphs. Recall that a **Hamiltonian cycle**  $\pi$  is a path in a graph  $G$  that visits each vertex precisely once before returning to the starting point. HC is  $NP$ -complete,

so a proof of knowledge for HC would provide a proof of knowledge for all  $NP$  problems: given any instance of a problem in  $NP$ , we can transform it into a graph with a Hamiltonian cycle if and only if the instance is a “Yes” instance. We can then use the HC proof for any  $NP$  problem.

A graph with  $n$  vertices can be represented by an  $n \times n$  binary matrix called the graph’s **adjacency matrix**. If the  $i$ th vertex is connected to the  $j$ th vertex, the  $ij$  entry in the matrix is 1; otherwise it is 0. Given a permutation  $\pi$  over  $\{1, \dots, n\}$  and a graph  $G$  defined by its adjacency matrix  $(a_{ij})$ , we define the permuted graph  $G^\pi$  as the graph that has the adjacency matrix  $(a'_{ij}) = (a_{\pi^{-1}(i)\pi^{-1}(j)})$ . A Hamiltonian cycle in a graph can in fact be represented by a permutation  $\pi$  of the vertices with the special property that the graph  $G^\pi$  includes the edges  $(1, 2), (2, 3), \dots, (n - 1, n), (n, 1)$ .

If  $\pi$  is a Hamiltonian cycle for a graph  $G$  and  $\pi'$  is an arbitrary permutation, then  $\pi'^{-1} \circ \pi$  is a Hamiltonian cycle for the graph  $G^{\pi'}$ .

HVZK proofs are used to verify that a Hamiltonian cycle corresponds to a given graph without revealing the cycle. Figure ?? demonstrates how such an HVZK proof is executed. Note that a dishonest prover can continue unnoticed with probability  $\kappa = 1/2$ . Suppose that a prover commits to a fake adjacency matrix such that she constructs a Hamiltonian cycle. If the verifier returns  $c = 1$ , the prover is able to convince the verifier that she knows a Hamiltonian cycle for the graph. But if the prover receives  $c = 0$ , the verifier learns that the prover did not commit to an accurate permutation of the graph. Through  $k$  repetitions however, we can decrease the knowledge error to  $\kappa = 1/2^k$  and thus prove the soundness property.

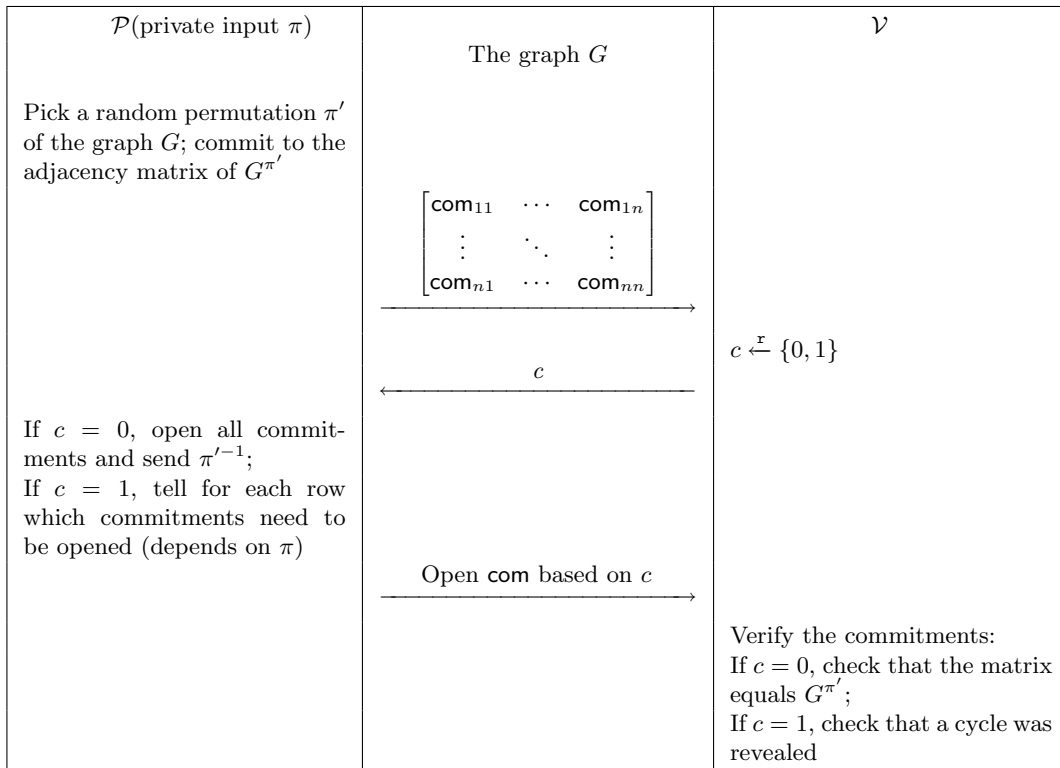


Figure 15: A proof of knowledge for a Hamiltonian cycle. In addition to committing to the adjacency matrix of  $G^{\pi'}$ , the prover must make a separate commitment  $\text{com}_{ij}$  to each entry in the matrix.

## 7.6 The Conjunction of Two Zero-Knowledge Proofs

There are instances in which a prover wants to validate multiple statements through one interaction, either for efficiency or privacy. This can be done using a single challenge value to preserve the desirable three-move structure. Upon receiving the challenge, the prover combines the offers and responses as is seen in Figure ??.

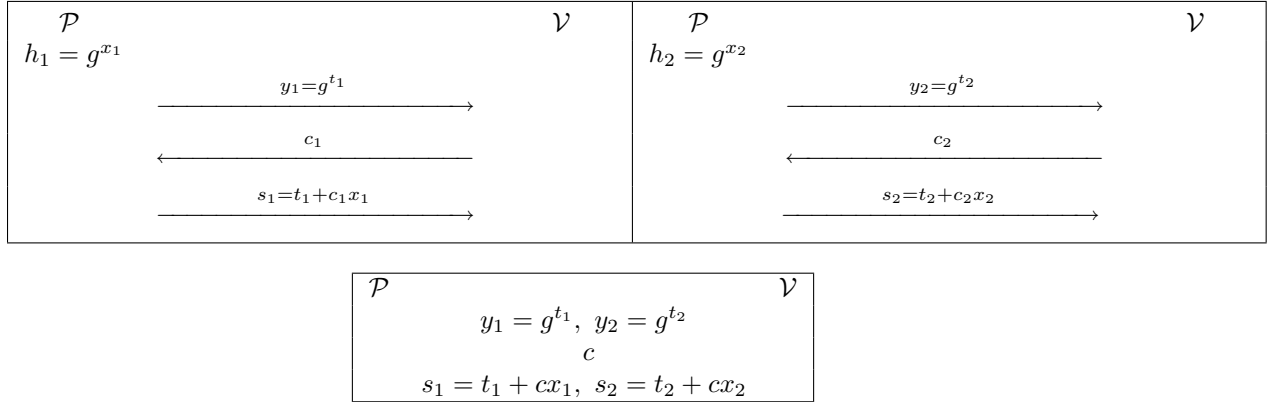


Figure 16: The conjunction of two zero-knowledge proofs for the discrete logarithm.

**Theorem 7.6.1.** *The conjunction of two honest-verifier zero-knowledge proofs satisfies the soundness and HVZK properties.*

## 7.7 The Disjunction of Two Zero-Knowledge Proofs

In Section ?? we mentioned that some user-identification schemes contain directories cataloging statements of theorems assigned to each user. In such schemes, privacy issues may arise when users wish to gain access without explicitly identifying themselves. In the disjunction of two zero-knowledge proofs, the identification protocol asks the user  $\mathcal{P}$  to provide witnesses to two statements. The user obviously knows a witness to one of the statements, but does not need to disclose which one. The system  $\mathcal{V}$  sends a single challenge value which the prover uses to fabricate a witness to the second statement. Figure ?? depicts the execution of a disjunction.

**Theorem 7.7.1.** *The disjunction of two honest-verifier zero-knowledge proofs satisfies the soundness and HVZK properties.*

## 8 Public-Key Encryption

In Section 3 we defined symmetric cryptosystems as encryption and decryption algorithms that share common key spaces. In an asymmetric cryptosystem, there are two distinct key spaces.

**Definition 8.0.1.** An *asymmetric cryptosystem* is composed of the the following elements:

- A plaintext message space  $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda \in \mathbb{N}}$
- A ciphertext message space  $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$
- A public key space  $\mathcal{P} = \{\mathcal{P}_\lambda\}_{\lambda \in \mathbb{N}}$  and a secret key space  $\mathcal{S} = \{\mathcal{S}_\lambda\}_{\lambda \in \mathbb{N}}$
- An efficient encryption algorithm  $\mathcal{E}: \mathcal{K}_p \times \mathcal{M} \rightarrow \mathcal{C}$  that preserves  $\lambda$ :  $\mathcal{E}(\mathcal{P}_\lambda \times \mathcal{M}_\lambda) \subseteq \mathcal{C}_\lambda$

$\mathcal{P}$		$\mathcal{V}$
$h_1 = g^{x_1}, h_2 = g^{x_2}$		
$y_1 = g^{t_1}$		
$s_2, c_2 \stackrel{\mathcal{R}}{\leftarrow} \mathbb{Z}_m$		
$y_2 = g^{s_2} h_2^{-c_2}$	$y_1, y_2$	
$c_1 = c - c_2 \pmod{m}$	$c$	
$s_1 = t_1 + c_1 x_1$	$c_1, c_2, s_1, s_2$	$c \stackrel{?}{=} c_1 + c_2$
		$g^{s_1} \stackrel{?}{=} y_1 h_1^{c_1}$
		$g^{s_2} \stackrel{?}{=} y_2 h_2^{c_2}$

Figure 17: The disjunction of two zero-knowledge proofs for the discrete logarithm showing how the prover can show the verifier he knows one of the two discrete logarithms of  $h_1, h_2$  (without revealing which one). In this case the prover  $\mathcal{P}$  knows a witness for  $h_1$ .

- An efficient decryption algorithm  $\mathcal{D}: \mathcal{K}_s \times \mathcal{C} \rightarrow \mathcal{M}$  that preserves  $\lambda$ :  $\mathcal{D}(\mathcal{S}_\lambda \times \mathcal{C}_\lambda) \subseteq \mathcal{M}_\lambda$
- An efficient key generation algorithm  $\mathcal{G}: \mathbb{N} \rightarrow \mathcal{P} \times \mathcal{S}$  that preserves  $\lambda$ :  $\mathcal{G}(1^\lambda)$ .

In addition to the above, an asymmetric cryptosystem must satisfy the *correctness property* that

- For all  $M \in \mathcal{M}$  and  $\langle pk, sk \rangle \in \mathcal{K}_p \times \mathcal{K}_s$ ,  $\mathcal{D}(sk, \mathcal{E}(pk, M)) = M$ .

## 8.1 AON-CPA Security

AON-CPA is one of the weakest security models for an asymmetric public-key cryptosystem. AON refers to the adversarial goal, “all-or-nothing”, where an attacker attempts to decrypt the ciphertext to obtain the plaintext. CPA corresponds to the adversarial capability, the chosen-plaintext attack. In a chosen-plaintext mode of attack, it is assumed that the attacker can obtain encryptions on messages of his choosing. Using these plaintext-ciphertext pairs, the attacker may be able to weaken the targeted system and experiment with the public-key encryption mechanism.

**Definition 8.1.1.** A public-key cryptosystem is AON-CPA secure provided

$$\text{Prob}[\mathcal{A}(pk, c) = M: \langle pk, sk \rangle \leftarrow \mathcal{G}(1^\lambda), c \leftarrow \mathcal{E}(pk, M), M \stackrel{\mathcal{R}}{\leftarrow} \mathcal{M}_\lambda]$$

is negligible in  $\lambda$  for all adversaries  $\mathcal{A}$ .

This model is weak in that it assumes the adversary wants to recover the entire plaintext. Moreover, such plaintext is random. AON-CPA does not prevent an attacker from recovering partial information.

## 8.2 IND-CPA Security

A stronger security model is IND-CPA. An adversary is allowed to submit two plaintext messages to the encryption oracle, which returns an encryption on one of the two plaintexts at random. The adversary must then discern which of the two plaintexts was returned. The adversarial goal IND stands for indistinguishability. We can model this by the following game,  $\text{Game}_{\text{IND-CPA}}^A(1^\lambda)$ .

1.  $\langle pk, sk \rangle \leftarrow \mathcal{G}(1^\lambda)$
2.  $\langle \text{aux}, M_0, M_1 \rangle \leftarrow \mathcal{A}(\text{play}, pk)$  for  $M_0 \neq M_1$



3.  $b \xleftarrow{r} \{0, 1\}$
4.  $c \leftarrow \mathcal{E}(pk, M_b)$
5.  $b^* \leftarrow \mathcal{A}(\text{guess}, \text{aux}, c)$
6. If  $b = b^*$  output 1; otherwise 0.

Figure ?? illustrates this game.

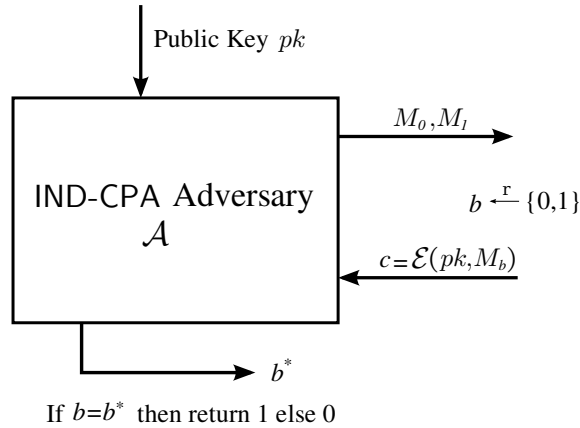


Figure 18: The IND-CPA attack. If  $b = b^*$ , we say the adversary wins the game.

**Definition 8.2.1.** A public-key cryptosystem is IND-CPA secure if for all PPT adversaries  $\mathcal{A}$ ,

$$\text{Prob}[\text{Game}_{\text{IND-CPA}}^{\mathcal{A}}(1^\lambda) = 1] \leq \frac{1}{2} + \text{negl}(\lambda).$$

**Example.** The RSA encryption scheme below is not IND-CPA secure. In fact, all deterministic public-key encryption schemes fail IND-CPA security.

$$\begin{aligned} \mathcal{G}(1^\lambda) : \quad & \langle pk, sk \rangle \leftarrow \mathcal{G}(1^\lambda) \\ & pk = (n, e) \\ & sk = d \end{aligned}$$

$$\begin{aligned} \mathcal{E}(pk, M) : \quad & M \in \{0, 1\}^\lambda \\ & \text{compute } c = M^e \bmod n \\ & \text{output } c \end{aligned}$$

$$\begin{aligned} \mathcal{D}(sk, c) : \quad & M = c^d \bmod n \\ & \text{output } M \end{aligned}$$

RSA encryption fails IND-CPA security because a fixed message is always encoded as the same ciphertext. Since the adversary has access to the encryption algorithm, he has the ability to view encryptions on both his messages independent of the system. When the oracle returns one of the encrypted messages, the adversary can reference his calculations to determine which message he was given. This implies that a scheme that is secure under IND-CPA must be a probabilistic protocol.

We can modify the RSA function to view it as a probabilistic scheme.

$$\begin{aligned}
\mathcal{G}(1^\lambda) : & \quad \langle pk, sk \rangle \leftarrow \mathcal{G}(1^\lambda) \\
& \quad pk = (n, e) \\
& \quad sk = d \\
\\
\mathcal{E}(pk, M) : & \quad M \in \{0, 1\}^{\lambda - \lambda_0} \\
& \quad r \in \{0, 1\}^{\lambda_0} \\
& \quad M' = \text{bit2integer}(r \| M) \\
& \quad \text{compute } c = M'^e \bmod n \\
& \quad \text{output } c \\
\\
\mathcal{D}(sk, c) : & \quad M' = c^d \bmod n \\
& \quad (r \| M) = \text{integer2bit}(M') \\
& \quad \text{output } M
\end{aligned}$$

By introducing the randomness  $r$ , the protocol no longer encrypts a message the same every time. This modification circumvents the problem found in the deterministic model. It is still uncertain however, if this probabilistic variant is susceptible to the IND-CPA attack.

### 8.3 ElGamal Encryption

ElGamal encryption is an asymmetric public-key algorithm that is provably secure in the IND-CPA model. It is based on the Diffie-Hellman key exchange, so it is defined over a cyclic group  $\langle g \rangle$  of prime order  $m$ .

$$\begin{aligned}
\mathcal{G}(1^\lambda) : & \quad \langle pk, sk \rangle \leftarrow \mathcal{G}(1^\lambda) \\
& \quad x \xleftarrow{\mathcal{R}} \mathbb{Z}_m, h = g^x \bmod p \\
& \quad pk = \langle \langle p, m, g \rangle, h \rangle \\
& \quad sk = x \\
\\
\mathcal{E}(pk, M) : & \quad M \in \langle g \rangle \\
& \quad r \xleftarrow{\mathcal{R}} \mathbb{Z}_m \\
& \quad \text{compute } G = g^r \bmod p, H = h^r M \bmod p \\
& \quad \text{output } \langle G, H \rangle \\
\\
\mathcal{D}(sk, G, H) : & \quad \text{compute } M = H/G^x \bmod p \\
& \quad \text{output } M
\end{aligned}$$

We can prove this is secure in the IND-CPA model under the DDH assumption. Given a PPT IND-CPA adversary  $\mathcal{B}$  such that

$$\text{Prob}[\text{Game}_{\text{IND-CPA}}^{\mathcal{B}}(1^\lambda) = 1] \geq \frac{1}{2} + \alpha$$

for a nonnegligible  $\alpha$ , we will construct a PPT algorithm  $\mathcal{A}$  that can distinguish the DDH tuple from the random tuple.

Algorithm  $\mathcal{A}(\langle p, m, g \rangle, h, G, H)$ :

1.  $\langle \text{aux}, M_0, M_1 \rangle \leftarrow \mathcal{B}(\text{play}, pk, h)$  where  $pk = \langle p, m, g \rangle$
2.  $b \xleftarrow{\mathcal{R}} \{0, 1\}$
3.  $c \leftarrow \langle G, H \cdot M_b \rangle$

4.  $b^* \leftarrow \mathcal{B}(\text{guess}, \text{aux}, c)$
5. If  $b = b^*$  output 1; otherwise 0.

Figure ?? illustrates how  $\mathcal{A}$  uses the attacker  $\mathcal{B}$  to break the DDH assumption.

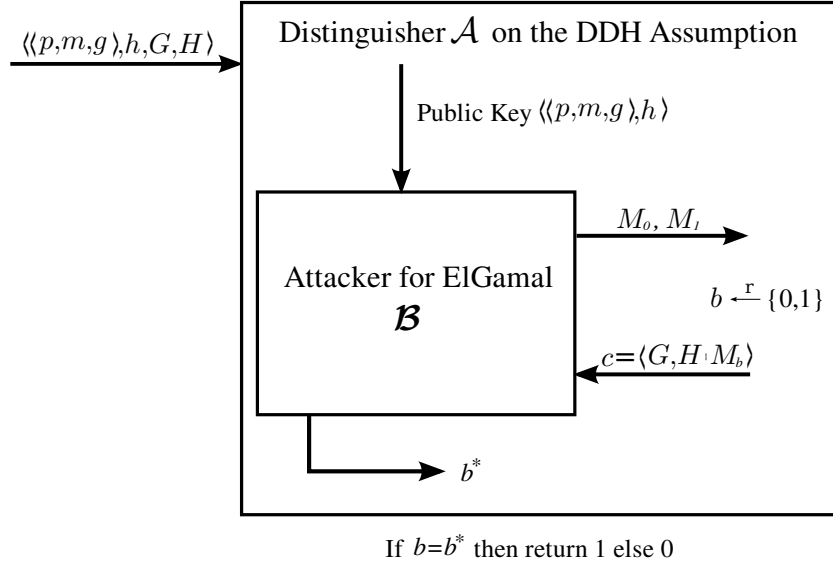


Figure 19: If an attacker  $\mathcal{B}$  decodes a message in ElGamal with a nonnegligible probability, we can construct a PPT distinguisher  $\mathcal{A}$  that breaks the DDH assumption. If  $b = b^*$ ,  $\mathcal{A}$  is successful.

Let  $v = \langle\langle p, m, g \rangle, g^x, g^y, g^{xy}\rangle$  be the DDH tuple. Then

$$\text{Prob}_{v \leftarrow \langle\langle p, m, g \rangle, g^x, g^y, g^{xy}\rangle} [\mathcal{A}(v) = 1] = \text{Prob}[\text{Game}_{\text{IND-CPA}}^{\mathcal{B}}(1^\lambda) = 1] \geq \frac{1}{2} + \alpha.$$

If  $v$  is the random tuple,  $v = \langle\langle p, m, g \rangle, g^x, g^y, g^z\rangle$ . For any  $w \in \langle g \rangle$ , we can find a unique  $z$  such that  $w = g^z M_b$  or  $z = \log_g(w/M_b)$ . Therefore no information about  $b$  is passed to the adversary. Since  $b^*$  is independent of the choice of  $b$ , the likelihood that  $b = b^*$  is  $1/2$ .

$$\text{Prob}_{v \leftarrow \langle\langle p, m, g \rangle, g^x, g^y, g^z\rangle} [\mathcal{A}(v) = 1] = \text{Prob}[b = b^*] = \frac{1}{2}.$$

Based on this,

$$\left| \text{Prob}_{v \leftarrow \langle\langle p, m, g \rangle, g^x, g^y, g^{xy}\rangle} [\mathcal{A}(v) = 1] - \text{Prob}_{v \leftarrow \langle\langle p, m, g \rangle, g^x, g^y, g^z\rangle} [\mathcal{A}(v) = 1] \right| \geq \alpha.$$

Because  $\alpha$  is nonnegligible, this violates the DDH assumption as desired.

## 9 Structuring Security Proofs as Sequences of Games

We now change gears to look at how we can structure security proofs as sequences of games. These games are best understood if read in parallel with the proofs in the subsequent sections.

## 9.1 Game Basics

**Lemma 9.1.1.** *For any  $I \subseteq \mathbb{N}$ , let  $\{a_i\}_{i \in I}, \{b_i\}_{i \in I}, \{c_i\}_{i \in I}$  be sequences of real numbers such that  $\sum_{i \in I} b_i = \sum_{i \in I} c_i = 1$  for all  $i \in I \subseteq \mathbb{N}$ . Then for all  $0 \leq a_i \leq 1$ , it holds that*

$$\left| \sum_{i \in I} a_i (b_i - c_i) \right| \leq \frac{1}{2} \sum_{i \in I} |b_i - c_i|.$$

**Definition 9.1.1.** Let  $G$  be a deterministic program that takes  $\lambda$ -bit inputs and terminates after  $v$  steps. At each step,  $G$  employs a random variable  $\rho_i$  sampled uniformly from the domain  $R_i$ . Each space  $R_i$  is parameterized by the  $\lambda$ -bit input and may depend on previous variables:  $R_i = \{0, 1\}^{s_i(\lambda)}$  for  $s_i: \mathbb{N} \rightarrow \mathbb{N}$  and  $1 \leq i \leq v$ . The  $i$ th step of  $G$  has the form

$$y \leftarrow f(y_1, \dots, y_d, \rho_1, \dots, \rho_i); \quad \rho_i \stackrel{\mathcal{F}}{\leftarrow} R_i^{y_1, \dots, y_d}$$

where  $y$  is a variable,  $y_1, \dots, y_d$  are variables from the  $i - 1$  step of  $G$ , and  $f$  is a deterministic polynomial-time computable function.

In addition to the  $v$  steps,  $G$  may contain other deterministic instructions. For example,  $G$  may employ conditional statements (if-then-else) and for-loops that cause a sequence of steps to repeat. In such instances, each repetition is counted as an additional step. No matter which statements  $G$  contains, the program structure can be depicted in a straight-line or tree-structure (based on the conditionals) sequence of assignments and function calls. We assume that the program returns a single bit as an outcome.

We call  $G$  a **game** and define  $T$  to be the winning event  $G$  outputs 1. (The probability space is comprised of all variables  $(\rho_1, \dots, \rho_v)$ ).

**Example.** Let PRIME be a probabilistic procedure that takes  $1^\lambda$  as an input, uses coin tosses in Coins, and returns a  $\lambda$ -bit prime as an output.

Game $G_0$ on Input $1^\lambda$	Random Variables
1. $p \leftarrow \text{PRIME}(1^\lambda, \rho)$	$\rho \stackrel{\mathcal{F}}{\leftarrow} \text{Coins}$
2. $x \leftarrow \sum_{i=0}^{\lambda-1} 2^i \rho_{i+1}$	$\rho_1, \dots, \rho_\lambda \stackrel{\mathcal{F}}{\leftarrow} \{0, 1\}$
3. output the least significant bit of $x \bmod p$	

## 9.2 The First Game-Playing Lemma

This first lemma allows us to connect the winning probabilities of two games that *proceed identically* unless a certain *failure event* happens. This lemma is used extensively in the existing literature employing games as a means to structure proofs. In our framework, we use Lemma ?? similar to [?]. That is, we refrain from using any special syntactical conditions and take into account the special syntax properties of the underlying game program.

**Lemma 9.2.1. (First Game-Playing Lemma).** *Consider two games,  $G$  and  $G'$  defined over the same random variables  $\rho_1, \dots, \rho_v$  in  $R_1, \dots, R_v$ . If  $\text{Prob}[T \cap \neg F] = \text{Prob}[T' \cap \neg F]$  for some event  $F$ , then  $|\text{Prob}[T] - \text{Prob}[T']| \leq \text{Prob}[F]$ .*

*Proof.* It holds that

$$\begin{aligned} \text{Prob}[T] &= \text{Prob}[T \cap F] + \text{Prob}[T \cap \neg F] \\ \text{Prob}[T'] &= \text{Prob}[T' \cap F] + \text{Prob}[T' \cap \neg F]. \end{aligned}$$

So by our assumption,

$$\text{Prob}[T] = \text{Prob}[T \cap F] + \text{Prob}[T' \cap \neg F].$$

It follows then that

$$|\text{Prob}[T] - \text{Prob}[T']| \leq |\text{Prob}[T \cap F] - \text{Prob}[T' \cap F]|.$$

Since the maximum distance between the probabilities on the right-hand side is  $\text{Prob}[F]$ , the result follows immediately.  $\blacksquare$

**Example.** Suppose we add another step to game  $G_0$  to obtain game  $G'_0$ .

Game $G'_0$ on Input $1^\lambda$	Random Variables
<ol style="list-style-type: none"> <li>1. <math>p \leftarrow \text{PRIME}(1^\lambda, \rho)</math></li> <li>2. <math>x \leftarrow \sum_{i=0}^{\lambda-1} 2^i \rho_{i+1}</math></li> <li>3. if <math>x &lt; 2^{\lfloor \lambda/2 \rfloor}</math> then <math>x \leftarrow 2x - p</math></li> <li>4. output the least significant bit of <math>x \bmod p</math></li> </ol>	$\rho \stackrel{\mathcal{R}}{\leftarrow}$ Coins $\rho_1, \dots, \rho_\lambda \stackrel{\mathcal{R}}{\leftarrow} \{0, 1\}$

Note that both games are defined over the same probability space; moreover, if  $F$  is the event  $x < 2^{\lfloor \lambda/2 \rfloor}$ , then the events  $T_0 \cap \neg F$  and  $T'_0 \cap \neg F$  are identical. Using the first game-playing lemma, we have that  $|\text{Prob}[T_0] - \text{Prob}[T'_0]| \leq \text{Prob}[F]$ . It is then easy to verify that  $\text{Prob}[F] \leq 2^{-\lfloor \lambda/2 \rfloor}$ .

### 9.3 The Second Game-Playing Lemma

The second game-playing lemma is a fundamental tool which we use pervasively in the following proofs. It is considered to be the basic “glue” in structuring proofs as sequences of games in our framework.

**Definition 9.3.1.** Suppose  $G$  is a game and the variables  $y_1, \dots, y_d$  are defined in  $G$ 's syntactic description up to the  $i$ th step. Let  $\tilde{y}_1, \dots, \tilde{y}_d$  be values in the respective domains of  $y_1, \dots, y_d$ . Let  $K$  be the event  $\bigwedge_{i=1}^d (y_i = \tilde{y}_i)$  and suppose  $K$  is a nonzero probability event in the probability space over which the game is defined. Define the **conditioned game**  $G[K]$  to be the game whose syntactic description is derived from that of  $G$  as follows:

1. The first  $i$  steps of  $G$  are omitted in  $G[K]$ .
2. All occurrences of the variables  $y_1, \dots, y_d$  are substituted by  $\tilde{y}_1, \dots, \tilde{y}_d$ .
3. An initial dummy step is added to  $G[K]$  that draws the random variables  $\rho_1, \dots, \rho_i$  from their respective spaces following their respective distributions conditioned on the event  $K$ .

**Example.** Consider the following game where  $\text{PRIME}$  is defined as before and  $f_1, f_2$  are two polynomial-time functions. The range of  $f_1(p, \cdot)$  is  $\mathbb{Z}_p^*$  and the range of  $f_2$  is  $\{0, 1\}$ .

Game $G_1$ on Input $1^\lambda$	Random Variables
<ol style="list-style-type: none"> <li>1. <math>p \leftarrow \text{PRIME}(1^\lambda, \rho)</math></li> <li>2. <math>x \leftarrow b + at \bmod p</math></li> <li>3. <math>c \leftarrow f_1(p, x; \rho_1)</math></li> <li>4. <math>y \leftarrow c + t \bmod p</math></li> <li>5. <math>b^* \leftarrow f_2(p, y; \rho_2)</math></li> <li>6. if <math>b = b^*</math> output 1</li> </ol>	$\rho \stackrel{\mathcal{R}}{\leftarrow}$ Coins $a \stackrel{\mathcal{R}}{\leftarrow} [p], t \stackrel{\mathcal{R}}{\leftarrow} [p-1], b \stackrel{\mathcal{R}}{\leftarrow} \{0, 1\}$ $\rho_1 \stackrel{\mathcal{R}}{\leftarrow}$ Coins <sub>1</sub> $\rho_2 \stackrel{\mathcal{R}}{\leftarrow}$ Coins <sub>2</sub>

We use  $[m]$  to denote the set  $\{1, \dots, m\}$  for  $m \in \mathbb{N}$ .

Now suppose  $p_0$  is a prime in the range of PRIME,  $x_0 \in \mathbb{Z}_p$ ,  $c_0 \in \mathbb{Z}_p^*$ , and  $y_0 \in \mathbb{Z}_p - \{c_0\}$ . We will condition the game  $G_1$  up to Step 4 based on these values. The conditioned game  $G_2 = G_1[p = p_0, x = x_0, c = c_0, y = y_0]$  is:

Game $G_2$ on Input $1^\lambda$	Random Variables
1.	$\langle a, t, b \rangle \stackrel{\mathcal{R}}{\leftarrow} R_{p_0, x_0, y_0}$
2. $b^* \leftarrow f_2(p_0, y_0; \rho_2)$	$\rho_2 \stackrel{\mathcal{R}}{\leftarrow} \text{Coins}_2$
3. if $b = b^*$ output 1	

The randomness space  $R_{p_0, x_0, y_0}$  contains the two triples  $\langle x_0(y_0 - c_0)^{-1}, y_0 - c_0, 0 \rangle$  and  $\langle (x_0 - 1)(y_0 - c_0)^{-1}, y_0 - c_0, 1 \rangle$ .

**Definition 9.3.2.** Let  $G$  be a game. A game  $G'$  is called *derivable* if it can be obtained from  $G$  using any number of the following modifications.

1.  $G'$  has the same description as  $G$  with one variable renamed.
2. Suppose  $\rho \stackrel{\mathcal{R}}{\leftarrow} R$  is a random variable drawn in some step of  $G$  and  $f(\rho)$  is a constant  $c$  for some function  $f$ . All occurrences of  $f(\rho)$  in  $G$  are substituted by  $c$  in  $G'$ .
3. Suppose  $\langle \rho_1, \rho_2 \rangle \stackrel{\mathcal{R}}{\leftarrow} R_1 \times R_2$  is one of the random variables drawn in some step of  $G$  so that  $\rho_1, \rho_2$  are independent and  $\rho_2$  is never used in  $G$ .  $G'$  is obtained by  $G$  by substituting  $\langle \rho_1, \rho_2 \rangle \stackrel{\mathcal{R}}{\leftarrow} R_1 \times R_2$  with  $\rho_1 \stackrel{\mathcal{R}}{\leftarrow} R_1$ .
4. Suppose a random variable  $\rho$  is drawn at Step  $i$ , but not used until Step  $j$  with  $j > i$ . Then  $\rho$  can be drawn at any Step  $1, \dots, j$ .

When  $G'$  is derived from  $G$ , we write  $G \stackrel{*}{\Rightarrow} G'$ . Two games  $G_1$  and  $G_2$  are called *joinable* if  $G_1 \stackrel{*}{\Rightarrow} G$  and  $G_2 \stackrel{*}{\Rightarrow} G$  for some game  $G$ .

**Proposition 9.3.1.** If  $G$  and  $G'$  are joinable,  $\text{Prob}[T] = \text{Prob}[T']$ .

The proof is simple and therefore omitted.

**Example.** Using Steps 3 and 4, we derive the following game from game  $G_2$ .

Game $G_3$ on Input $1^\lambda$ ( $[p_0, x_0, c_0, y_0]$ are constants)	Random Variables
1. $b^* \leftarrow f_2(p_0, y_0; \rho_2)$	$\rho_2 \stackrel{\mathcal{R}}{\leftarrow} \text{Coins}_2$
2. if $b = b^*$ output 1	$b \stackrel{\mathcal{R}}{\leftarrow} \{0, 1\}$

We now proceed to the second game-playing lemma. This is a syntax-based lemma that provides a substantial amount of freedom when modifying the steps in a game. Informally, the second game playing lemma allows us to change an instruction  $y \leftarrow f(\rho)$  to another instruction  $y' \leftarrow f'(\rho')$ . Inso doing we incur a distance bounded by the statistical distance of  $f$  and  $f'$  in the winning probability of the two games. In addition, we can modify the probability distribution of  $\rho'$  and select it differently from the distribution of  $\rho$ . This requires some care due primarily to the fact that the subsequent steps may depend on both  $y$  and the random variables used when defining  $y$ . For this reason, a *normalizing function norm* is used in the modified game to substitute in for any occurrence of  $\rho$ . The random variables  $\rho$  and  $\text{norm}(\rho)$  are assumed to be identically distributed. The exact formulation of the lemma is as follows.

**Lemma 9.3.1. (Second Game-Playing Lemma).** Consider a game  $G$  that contains the step

$$y \leftarrow f(y_1, \dots, y_d, \rho_1, \dots, \rho_i); \quad \rho_i \stackrel{\mathcal{R}}{\leftarrow} R_i^{y_1, \dots, y_d}.$$

Modify this step as follows to obtain the modified game  $G'$ :

$$y' \leftarrow f'(y_1, \dots, y_d, \rho_1, \dots, \rho_{i-1}, \rho'_i); \quad t \leftarrow \text{norm}(y_1, \dots, y_d, \rho_1, \dots, \rho_{i-1}, \rho'_i); \quad \rho'_i \stackrel{\mathcal{R}}{\leftarrow} (R_i^{y_1, \dots, y_d})'.$$

The two games proceed identically after the  $i$ th step with every reference of  $y$  in game  $G$  substituted with  $y'$  in game  $G'$ , and similarly for  $\rho_i$  substituted with  $t$ .<sup>6</sup> Let  $K$  be any event that fixes all variables in the first  $i - 1$  steps of both  $G$  and  $G'$ . Let  $D_y$  be the range of  $f$  as it is induced by the event  $K$  in game  $G$  and let  $D_{y'}$  be the range of  $f'$  as it is induced by the event  $K$  in game  $G'$ . Suppose that

1. For any such  $K$  and any  $y_0 \in D_y \cap D_{y'}$ , the conditioned games  $G[K \wedge (y = y_0)]$  and  $G'[K \wedge (y' = y_0)]$  are joinable.
2. For any  $K$ , the statistical distance of the probability distribution of  $y$  and  $y'$  over the extended support set  $D_y \cup D_{y'}$  conditioned on  $K$  is at most  $\varepsilon$ , where  $\varepsilon$  is a function in  $\lambda$  (independent of the choice of  $K$ ).

Given these two conditions,  $|\text{Prob}[T] - \text{Prob}[T']| \leq \varepsilon$ .

*Proof.* Let  $K$  be an event that fixes all variables in the first  $i - 1$  steps. Consider the ranges of the variables  $y$  and  $y'$ , denoted by  $D_y$  and  $D_{y'}$ , respectively, which are induced by the properties of the functions  $f$  and  $f'$  conditioned on the event  $K$ . Due to the equivalence of the two conditioned games, we derive that

$$\text{Prob}[T \mid K \wedge (y = y_0)] = \text{Prob}[T' \mid K \wedge (y' = y_0)].$$

Looking at  $\text{Prob}[T \mid K]$  and  $\text{Prob}[T' \mid K]$ , we see

$$\begin{aligned} \text{Prob}[T \mid K] &= \sum_{y_0 \in D_y} \text{Prob}[T \mid K \wedge (y = y_0)] \cdot \text{Prob}[y = y_0 \mid K] \\ &= \sum_{y_0 \in D_y \setminus D_{y'}} \text{Prob}[T \mid K \wedge (y = y_0)] \cdot \text{Prob}[y = y_0 \mid K] + \dots \\ &\quad \dots + \sum_{y_0 \in D_y \cap D_{y'}} \text{Prob}[T' \mid K \wedge (y' = y_0)] \cdot \text{Prob}[y = y_0 \mid K] \end{aligned}$$

and

$$\begin{aligned} \text{Prob}[T' \mid K] &= \sum_{y_0 \in D_{y'}} \text{Prob}[T' \mid K \wedge (y' = y_0)] \cdot \text{Prob}[y' = y_0 \mid K] \\ &= \sum_{y_0 \in D_{y'} \setminus D_y} \text{Prob}[T' \mid K \wedge (y' = y_0)] \cdot \text{Prob}[y' = y_0 \mid K] + \dots \\ &\quad \dots + \sum_{y_0 \in D_y \cap D_{y'}} \text{Prob}[T' \mid K \wedge (y' = y_0)] \cdot \text{Prob}[y' = y_0 \mid K]. \end{aligned}$$

Note that  $\text{Prob}[y' = y_0 \mid K] = 0$  if  $y_0 \in D_y \setminus D_{y'}$  and  $\text{Prob}[y = y_0 \mid K] = 0$  if  $y_0 \in D_{y'} \setminus D_y$ . By subtracting the two probabilities we find

$$|\text{Prob}[T \mid K] - \text{Prob}[T' \mid K]| = \left| \sum_{y_0 \in D_y \cup D_{y'}} C(y_0) (\text{Prob}[y = y_0 \mid K] - \text{Prob}[y' = y_0 \mid K]) \right|,$$

<sup>6</sup>It may not be necessary to substitute  $\rho_i$  with  $t$ . For example, we omit the instruction  $t \leftarrow (y_1, \dots, y_d, \rho_1, \dots, \rho_{i-1}, \rho'_i)$  from the modified game when  $\rho_i$  does not occur in the game, when  $\rho_i$  and  $\rho'_i$  are identically distributed, and when  $\rho'_i = \langle \rho_i, s \rangle$  for any  $s$ .

where the function  $C(y_0)$  is defined by

$$C(y_0) = \begin{cases} \text{Prob}[T' \mid K \wedge (y' = y_0)], & \text{if } y_0 \in D_{y'} \\ \text{Prob}[T \mid K \wedge (y = y_0)], & \text{if } y_0 \in D_y \setminus D_{y'} \end{cases}.$$

For all  $y_0 \in D_y \cup D_{y'}$ , it holds that  $0 \leq C(y_0) \leq 1$ . Using Lemma ??, we can then conclude

$$|\text{Prob}[T \mid K] - \text{Prob}[T' \mid K]| \leq \frac{1}{2} \sum_{y_0 \in D_y \cup D_{y'}} |\text{Prob}[y = y_0 \mid K] - \text{Prob}[y' = y_0 \mid K]| \leq \varepsilon$$

Now

$$\begin{aligned} |\text{Prob}[T] - \text{Prob}[T']| &= \left| \sum_K \text{Prob}[T \mid K] \cdot \text{Prob}[K] - \sum_K \text{Prob}[T' \mid K] \cdot \text{Prob}[K] \right| \\ &\leq \sum_K |\text{Prob}[T \mid K] - \text{Prob}[T' \mid K]| \cdot \text{Prob}[K] \\ &\leq \varepsilon \sum_K \text{Prob}[K] \\ &= \varepsilon. \end{aligned}$$

as desired. ■

**Example.** We can alter the choice of  $y$  in Step 4 of game  $G_1$  to obtain  $G_4$ .

Game $G_4$ on Input $1^\lambda$	Random Variables
1. $p \leftarrow \text{PRIME}(1^\lambda, \rho)$	$\rho \stackrel{\mathcal{R}}{\leftarrow} \text{Coins}$
2. $x \leftarrow b + at \pmod p$	$a \stackrel{\mathcal{R}}{\leftarrow} [p], t \stackrel{\mathcal{R}}{\leftarrow} [p-1], b \stackrel{\mathcal{R}}{\leftarrow} \{0, 1\}$
3. $c \leftarrow f_1(p, x; \rho_1)$	$\rho_1 \stackrel{\mathcal{R}}{\leftarrow} \text{Coins}_1$
4. $y \leftarrow Y$	$Y \stackrel{\mathcal{R}}{\leftarrow} [p]$
5. $b^* \leftarrow f_2(p, y; \rho_2)$	$\rho_2 \stackrel{\mathcal{R}}{\leftarrow} \text{Coins}_2$
6. if $b = b^*$ output 1	

Notice that in Games  $G_1, G_2$ , and  $G_3$ ,  $y$  contains information about  $b$ . By modifying Step 4, we break the chain to  $b$ , so  $b^*$  is now purely a random guess. We can use the second game-playing lemma to show that the winning probabilities of  $G_1$  and  $G_4$  are still very close. To accomplish this, we first consider  $G_1$  and  $G_4$  based on the conditioning  $p = p_0, x = x_0, c = c_0, y = y_0$ , where  $p_0$  is a prime number,  $x_0 \in \mathbb{Z}_p, c_0 \in \mathbb{Z}_p^*$ , and  $y_0 \in \mathbb{Z}_p - \{c_0\}$ .

Under this conditioning, game  $G_1$  is exactly game  $G_2$  and  $G_4$  produces the following game  $G_5$ .

Game $G_5$ on Input $1^\lambda$ ( $[p_0, x_0, c_0, y_0]$ are constants)	Random Variables
1.	$\langle a, t, b \rangle \stackrel{\mathcal{R}}{\leftarrow} R_{p_0, x_0, y_0}$
2. $b^* \leftarrow f_2(p_0, y_0; \rho_2)$	$\rho_2 \stackrel{\mathcal{R}}{\leftarrow} \text{Coins}_2$
3. if $b = b^*$ output 1	

The space  $R_{p_0, x_0, y_0}$  contains triples of the form  $\langle (x_0 - b)t^{-1}, t, b \rangle$ , where  $t \in [p-1]$  and  $b \in \{0, 1\}$ . Since  $a$  and  $t$  are not used in  $G_5$ , we can derive game  $G_3$  from game  $G_5$ ; thus  $G_2$  and  $G_5$  are joinable.

Now consider the conditional probability distributions of the variable  $y$  in  $G_2$  and  $G_5$ . By conditioning on  $p_0, x_0, c_0$  in game  $G_5$ , the variable  $y$  is uniformly distributed over  $[p_0]$ . On the other hand, the variable  $y$  equals  $c_0 + t$  when conditioning on  $p_0, x_0, c_0$  in game  $G_1$  to get  $G_2$ , where  $t$  is selected from the conditional distribution  $\langle (x_0 - b)t^{-1}, t, b \rangle$  for  $t \in [p-1]$  and  $b \in \{0, 1\}$ . Then all



values of  $[p]$ , except  $c_0$ , are equally likely with probability  $1/p - 1$ . This implies that the statistical distance between  $G_2$  and  $G_5$ .

$$\frac{1}{2} \left[ (p-1) \left( \frac{1}{p-1} - \frac{1}{p} \right) + \frac{1}{p} \right] = \frac{1}{p} \leq 2^{-\lambda+1}.$$

The second game playing lemma gives  $|\text{Prob}[T_1] - \text{Prob}[T_4]| \leq 2^{-\lambda+1}$ .

## 9.4 The Third Game-Playing Lemma

**Lemma 9.4.1. (Third Game-Playing Lemma).** *Let  $\mathcal{D}_1^\lambda$  and  $\mathcal{D}_2^\lambda$  be two probability ensembles with  $\lambda \in \mathbb{N}$ , such that for all PPT  $\mathcal{A}$ ,*

$$\left| \text{Prob}_{\vec{y} \leftarrow \mathcal{D}_1^\lambda} [\mathcal{A}(1^\lambda, \vec{y}) = 1] - \text{Prob}_{\vec{y} \leftarrow \mathcal{D}_2^\lambda} [\mathcal{A}(1^\lambda, \vec{y}) = 1] \right| \leq \delta(\lambda).$$

*If  $G_1$  and  $G_2$  are two otherwise identical games such that whenever a sequence of variables  $\vec{y} = \langle y_1, \dots, y_d \rangle$  is distributed according to  $\mathcal{D}_1^\lambda$  in game  $G_1$ , the same sequence of variables is distributed according to  $\mathcal{D}_2^\lambda$  in game  $G_2$ ; moreover, no random variable used in defining  $\vec{y}$  is (directly) employed anywhere else in the game, then  $|\text{Prob}[T_1] - \text{Prob}[T_2]| \leq \delta(\lambda)$ .*

*Proof.* Using the fact that  $G_1$  and  $G_2$  are identical except for the choice of  $\vec{y}$ , we can build a PPT distinguisher for  $\mathcal{D}_1^\lambda, \mathcal{D}_2^\lambda$  that operates the same as the two games when given  $\vec{y}$ , except instead of sampling  $\vec{y}$  according to the stated instructions, it employs its input vector  $\vec{y}$ . This distinguisher will behave either as game  $G_1$  (if  $\vec{y}$  is drawn from  $\mathcal{D}_1^\lambda$ ) or as game  $G_2$  (if  $\vec{y}$  is drawn from  $\mathcal{D}_2^\lambda$ ). Note that it is critical to assume no game uses any random variable employed in the choice of  $\vec{y}$  to ensure the distinguisher's design is independent of  $\vec{y}$ .  $\blacksquare$

**Notation.** For any PPT  $\mathcal{A}$  with input of size  $\lambda$ , we assume  $\mathcal{A}$  always uses  $p(\lambda)$  coin tosses, where  $p$  is a fixed polynomial (dependent only on  $\mathcal{A}$ ). We write  $\mathcal{A}(x; \rho)$  to simulate  $\mathcal{A}$  deterministically with the string  $\rho \in \{0, 1\}^{p(\lambda)}$  as the coin tosses.

## 10 PRPs versus PRFs

Here we analyze the pseudorandom permutation/pseudorandom function (PRP/PRF) lemma discussed in [?]. The essential idea of the proof is the same as in the previous examples (after all, proofs are proofs and games are games), but the presentation of the proof is quite different using the present methodology. We will invoke our second game-playing lemma instead of the first.

Our goal is to show that for all PPT predicates  $\mathcal{A}$ ,

$$|\text{Prob}[\mathcal{A}^f(1^\lambda) = 1] - \text{Prob}[\mathcal{A}^\pi(1^\lambda) = 1]| = \text{negl}(\lambda),$$

where  $f$  is drawn at random from all functions  $\{0, 1\}^\lambda \rightarrow \{0, 1\}^\lambda$  and  $\pi$  is drawn at random from all permutations  $\{0, 1\}^\lambda \rightarrow \{0, 1\}^\lambda$ .

Without loss of generality, assume: (1)  $\mathcal{A}$  never makes the same oracle query twice and (2)  $\mathcal{A}$  always makes  $Q$  oracle queries, where  $Q$  is a function in  $\lambda$ . We are allowed to make these two assumptions because without changing the functionality, we can transform any PPT predicate not adhering to these facts into a predicate that does.

Let PERM be a procedure that produces a random permutation  $\{0, 1\}^\lambda \rightarrow \{0, 1\}^\lambda$  on input  $1^\lambda$ . Suppose  $\mathcal{A}$  is a PPT predicate as specified above that uses  $t(\lambda)$  coin tosses. Consider the following game:

Game $G_1$ on Input $1^\lambda$	Random Variables
1. $\pi \leftarrow \text{PERM}(1^\lambda; \rho)$	$\rho \stackrel{\text{r}}{\leftarrow} \text{Coins}$
2. $b \leftarrow \mathcal{A}^\pi(1^\lambda; \rho')$	$\rho' \stackrel{\text{r}}{\leftarrow} \{0, 1\}^{t(\lambda)}$
3. output $b$	

The winning probability of game  $G_1$  equals  $\text{Prob}[\mathcal{A}^\pi(1^\lambda) = 1]$ . Suppose  $\mathcal{A}_0, \dots, \mathcal{A}_Q$  is a sequence of PPT algorithms defined by  $\mathcal{A}$  by splitting  $\mathcal{A}$  into the  $Q + 1$  stages around its oracle queries. This forms game  $G_2$ .

Game $G_2$ on Input $1^\lambda$	Random Variables
1. $\pi \leftarrow \text{PERM}(1^\lambda, \rho)$	$\rho \stackrel{\text{r}}{\leftarrow} \text{Coins}$
2. $\langle \text{aux}, i_1 \rangle \leftarrow \mathcal{A}_0(1^\lambda; \rho_1)$	$\rho_1 \stackrel{\text{r}}{\leftarrow} \{0, 1\}^\lambda$
3. for $j = 1$ to $Q - 1$	
4. $\mathbf{a}_j \leftarrow \pi(i_j)$	
5. $\langle \text{aux}, i_j \rangle \leftarrow \mathcal{A}_j(1^\lambda, \text{aux}, \mathbf{a}_j; \rho_j)$	$\rho_j \stackrel{\text{r}}{\leftarrow} \{0, 1\}^\lambda$
6. $\mathbf{a}_Q \leftarrow \pi(i_Q)$	
7. $b \leftarrow \mathcal{A}_Q(1^\lambda, \text{aux}, \mathbf{a}_Q; \rho_Q)$	$\rho_Q \stackrel{\text{r}}{\leftarrow} \{0, 1\}^\lambda$
8. output $b$	

Instead of sampling the permutation in the first statement, we can construct the random permutation “on the fly” using “lazy sampling” to modify  $G_2$ . This produces game  $G_3$ .

Game $G_3$ on Input $1^\lambda$	Random Variables
1. $\langle \text{aux}, i_1 \rangle \leftarrow \mathcal{A}_0(1^\lambda; \rho_1)$	$\rho_1 \stackrel{\text{r}}{\leftarrow} \{0, 1\}^\lambda$
2. for $j = 1$ to $Q - 1$	
3. $\mathbf{a}_j \leftarrow v_j$	$v_j \stackrel{\text{r}}{\leftarrow} R_\lambda^{\mathbf{a}_1 \dots \mathbf{a}_{j-1}}$
4. $\langle \text{aux}, i_{j+1} \rangle \leftarrow \mathcal{A}_j(1^\lambda, \text{aux}, \mathbf{a}_j; \rho_j)$	$\rho_j \stackrel{\text{r}}{\leftarrow} \{0, 1\}^\lambda$
5. $\mathbf{a}_Q \leftarrow v$	$v \stackrel{\text{r}}{\leftarrow} R^{\mathbf{a}_1, \dots, \mathbf{a}_{Q-1}}$
6. $b \leftarrow \mathcal{A}_Q(1^\lambda, \text{aux}, \mathbf{a}_Q; \rho_Q)$	$\rho_Q \stackrel{\text{r}}{\leftarrow} \{0, 1\}^\lambda$
7. output $b$	

Note that  $R_\lambda^{\mathbf{a}_1 \dots \mathbf{a}_k} = \{0, 1\}^\lambda - \{\mathbf{a}_1, \dots, \mathbf{a}_k\}$

**Modification [3,j].** We modify the 3rd step during the  $j$ th execution of the for-loop so that it reads like this (this actually requires splitting the single for-loop of  $G_2$  into two for-loops, but we refrain from writing this):

3. $\mathbf{a}_j \leftarrow v_j$	$v_j \stackrel{\text{r}}{\leftarrow} \{0, 1\}^\lambda$
----------------------------------	--

When conditioning on any choice of pairwise distinct  $\mathbf{a}_1, \dots, \mathbf{a}_{j-1}$ ; any  $i_1, \dots, i_j$ ; and any  $\text{aux}_1, \dots, \text{aux}_j$ , it holds that the probability distribution of  $\mathbf{a}_j$  in game  $G_{3,j}$  is uniform over  $\{0, 1\}^\lambda$ . On the other hand, the probability distribution of  $\mathbf{a}_j$  in game  $G_{3,j-1}$  is uniform over  $\{0, 1\}^\lambda - \{\mathbf{a}_1, \dots, \mathbf{a}_{j-1}\}$ , i.e., each element will have probability  $1/(2^\lambda - j + 1)$  of succeeding. The statistical distance of the two conditional distributions is

$$\frac{1}{2} \left[ (2^\lambda - j + 1) \left( \frac{1}{2^\lambda - j + 1} - \frac{1}{2^\lambda} \right) + (j - 1) \left( \frac{1}{2^\lambda} \right) \right] = \frac{j - 1}{2^\lambda}.$$

Finally observe that the conditioned games  $G_{3,j-1}$  and  $G_{3,j}$  are equivalent. As a result, we can apply the second game playing lemma to obtain  $|\text{Prob}[T_{3,j-1}] - \text{Prob}[T_{3,j}]| \leq (j - 1)/2^\lambda$ . From this, we can deduce

$$|\text{Prob}[T_3] - \text{Prob}[T_{3,Q}]| \leq \sum_{j=1}^Q \frac{j-1}{2^\lambda} = \frac{Q(Q-1)}{2^{\lambda+1}}.$$

Game  $G_{3,Q}$  now has the form

Game $G_{3,Q}$ on Input $1^\lambda$	Random Variables
1. $\langle \text{aux}, i_1 \rangle \leftarrow \mathcal{A}_0(1^\lambda; \rho_1)$	$\rho_1 \xleftarrow{\mathcal{R}} \{0, 1\}^\lambda$
2. for $j = 1$ to $Q - 1$	
3. $\mathbf{a}_j \leftarrow v_j$	$v_j \xleftarrow{\mathcal{R}} \{0, 1\}^\lambda$
4. $\langle \text{aux}, i_{j+1} \rangle \leftarrow \mathcal{A}_j(1^\lambda, \text{aux}, \mathbf{a}_j; \rho_j)$	$\rho_j \xleftarrow{\mathcal{R}} \{0, 1\}^\lambda$
5. $\mathbf{a}_Q \leftarrow v$	$v \xleftarrow{\mathcal{R}} \{0, 1\}^\lambda$
6. $b \leftarrow \mathcal{A}_Q(1^\lambda, \text{aux}, \mathbf{a}_Q; \rho_Q)$	$\rho_Q \xleftarrow{\mathcal{R}} \{0, 1\}^\lambda$
7. output $b$	

Then game  $G_3$  can be “folded back” to game  $G_4$  with the identical operation:

Game $G_4$ on Input $1^\lambda$	Random Variables
1. $f \leftarrow \text{FUNC}(1^\lambda; \rho)$	$\rho \xleftarrow{\mathcal{R}} \text{Coins}$
2. $b \leftarrow \mathcal{A}^f(1^\lambda; \rho')$	$\rho' \xleftarrow{\mathcal{R}} \{0, 1\}^{t(\lambda)}$
3. output $b$	

**FUNC** is a procedure that samples a random function  $\{0, 1\}^\lambda \rightarrow \{0, 1\}^\lambda$ . The winning probability of  $G_4$  is exactly  $\text{Prob}[\mathcal{A}^f(1^\lambda) = 1]$ , where  $f$  is a random function  $\{0, 1\}^\lambda \rightarrow \{0, 1\}^\lambda$ . As a result, we obtain

$$|\text{Prob}[\mathcal{A}^f(1^\lambda) = 1] - \text{Prob}[\mathcal{A}^\pi(1^\lambda) = 1]| = \frac{Q(Q-1)}{2^{\lambda+1}},$$

which is negligible in  $\lambda$  since  $Q$  is assumed to be polynomial in  $\lambda$ .

## 11 The Cramer-Shoup Cryptosystem

Now that we have the basic foundation for proving security based on sequences of games, we want to prove that the Cramer-Shoup (CS) public-key cryptosystem is IND-CCA2 secure, where we define CS later in Section ???. The adversarial goal, CCA2 stands for the chosen-ciphertext attack with two rounds of decryption queries. Before proving IND-CCA2 security, we will prove several weaker results on simpler cryptosystems in order to build up to CS.

### 11.1 Step 1: Proving IND-CPA Security

Let  $\text{GGen}$  check be a group generator that when given a coin toss  $\rho$  and length  $1^\lambda$ , it produces  $\langle p, q, g_1 \rangle$  such that  $p$  is a  $\lambda$ -bit prime,  $q$  is an  $s(\lambda)$ -bit prime where  $s: \mathbb{N} \rightarrow \mathbb{N}$  is a given function, and  $g_1$  is an order  $q$  element in  $\mathbb{Z}_p^*$ . We assume  $s$  is selected so that  $2^{-s(\lambda)}$  is negligible in  $\lambda$ .

**Assumption.** When given a random instance of  $\langle p, q, g_1 \rangle$  from  $\text{GGen}(1^\lambda)$ , the advantage  $\text{Adv}_{\text{DDH}}^{\text{GGen}}$  of any PPT  $\mathcal{A}$  is negligible in  $\lambda$ . That is,  $\mathcal{A}$  can distinguish triples of the form  $\langle g_1^x, g_1^y, g_1^{xy} \rangle$  from triples of the form  $\langle g_1^x, g_1^y, g_1^z \rangle$  with negligible probability when  $x, y, z \xleftarrow{\mathcal{R}} [q]$ . This is the Decisional Diffie-Hellman assumption.

#### 11.1.1 The Two-Generator ElGamal Public-Key Cryptosystem

The two-generator variant of the ElGamal public-key encryption scheme is as follows:

**Key-generation.** For input  $1^\lambda$ , select a  $\lambda$ -bit prime  $p$  such that  $g_1$  is a prime order  $q$  element<sup>7</sup>, where  $q$  divides  $p - 1$  and  $q$  is  $s(\lambda) < \lambda$  bits. Select two random values  $w, z \xleftarrow{\mathcal{R}} [q]$ . The public key is  $\langle p, q, g_1, g_2, h \rangle$ , where  $g_2 \leftarrow g_1^w, h \leftarrow g_1^{z_1} g_2^{z_2}$ , and the secret key is  $z_1, z_2$ .

<sup>7</sup> $g_1$  is an element of a finite group of prime order  $q$ .

**Encryption.** The encryption function is given  $M \in \langle g_1 \rangle$  (we are not concerned with how “real” messages can be mapped to  $\langle g_1 \rangle$ ), samples  $r \xleftarrow{\mathcal{R}} [q]$ , and returns  $\langle u_1, u_2, v \rangle = \langle g_1^r, g_2^r, h^r M \rangle$ .

**Decryption.** When given a ciphertext  $\langle u_1, u_2, v \rangle$ , the decryption function returns  $v/u_1^{z_1} u_2^{z_2} \bmod p$ .

### The IND-CPA Game

Game $G_0$ on Input $1^\lambda$	Random Variables
1. $\langle p, q, g_1 \rangle \leftarrow \text{GGen}(1^\lambda; \rho)$	$\rho \leftarrow \text{Coins}$
2. $g_2 \leftarrow g_1^w$	$w \xleftarrow{\mathcal{R}} [q]$
3. $h \leftarrow g_1^{z_1} g_2^{z_2}$	$z_1, z_2 \xleftarrow{\mathcal{R}} [q]$
4. $\langle \text{aux}, M_0, M_1 \rangle \leftarrow \mathcal{A}(\text{stage}_1, g_1, g_2, h; \rho_1)$	$\rho_1 \xleftarrow{\mathcal{R}} \text{Coins}_1$
5. $u_1^* \leftarrow g_1^{r_1}; u_2^* \leftarrow g_2^{r_2}$	$r \xleftarrow{\mathcal{R}} [q]$
6. $v^* \leftarrow h^r M_b$	$b \xleftarrow{\mathcal{R}} \{0, 1\}$
7. $b^* \leftarrow \mathcal{A}(\text{stage}_2, \text{aux}, u_1^*, u_2^*, v^*; \rho_2)$	$\rho_2 \xleftarrow{\mathcal{R}} \text{Coins}_2$
8. if $b = b^*$ return 1 else return 0	

### Proof of Security

We present the proof of security structured as a sequence of games.

**Modification 1.** We modify Step 6 of game  $G_0$  to obtain game  $G_1$ .

6. $v^* \leftarrow (u_1^*)^{z_1} (u_2^*)^{z_2} M_b$	$b \xleftarrow{\mathcal{R}} \{0, 1\}$
---	---------------------------------------

This is merely a syntactic modification since  $(u_1^*)^{z_1} (u_2^*)^{z_2} M_b = h^r M_b$ ; thus, it follows immediately that  $\text{Prob}[T_0] = \text{Prob}[T_1]$ .

**Modification 2.** By modifying Step 5 of game  $G_1$ , we obtain game  $G_2$ .

5. $u_1^* \leftarrow g_1^{r_1}; u_2^* \leftarrow g_2^{r'_2}$	$r \xleftarrow{\mathcal{R}} [q], r' \xleftarrow{\mathcal{R}} [q]$
--	---

Note that the variables  $w, r, r'$  are not explicitly used in either  $G_1$  or  $G_2$ . Moreover, for any group description  $\langle p, q, g_1 \rangle$  produced by  $\text{GGen}$ , the triple  $\langle g_2, u_1^*, u_2^* \rangle$  is distributed as a DDH triple in game  $G_1$  and as a random triple drawn from  $\langle g_1 \rangle$  in game  $G_2$ . It follows from the third game-playing lemma that  $|\text{Prob}[T_1] - \text{Prob}[T_2]| \leq \text{Adv}_{\text{DDH}}^{\text{GGen}}(\lambda)$ .

**Modification 3.** We perform the following modification to game  $G_2$  to obtain game  $G_3$ :

2. $g_2 \leftarrow g_1^w$	$w \xleftarrow{\mathcal{R}} [q-1]$
---------------------------	------------------------------------

Conditioned on any choice of  $p, q, g_1$ , the statistical distance between the probability distribution of  $[g_2]_2$  and  $[g_2]_3$  is  $1/q$ ; indeed, we have

$$\frac{1}{2} \left[ (q-1) \left| \frac{1}{q} - \frac{1}{q-1} \right| + \frac{1}{q} \right] = \frac{1}{q} \leq \frac{1}{2^{s(\lambda)-1}}.$$

The variable  $w$  is used only through  $g_2$  in  $G_2$  and  $G_3$ , so it is easy to verify that game  $G_2$ , when conditioned on any choice of  $p, q, g_1$  and  $[g_2]_2 = \alpha$ , is equivalent to the conditioned game  $G_3$  on the same  $p, q, g_1$  and  $[g_2]_3 = \alpha$ . We conclude from the second game-playing lemma that  $|\text{Prob}[T_2] - \text{Prob}[T_3]| \leq 2^{-s(\lambda)+1}$ .

**Modification 4.** We perform the following modification to game  $G_3$  to obtain game  $G_4$ :

5. $u_1^* \leftarrow g_1^{r_1}; u_2^* \leftarrow g_2^{r'_2}$	$\langle r, r' \rangle \xleftarrow{\mathcal{R}} ([q] \times [q])^\neq$
--	--

where  $([q] \times [q])^\neq$  is the set of all pairs of distinct elements of  $[q]$ .

Suppose we condition on a choice of  $p, q, g_1, g_2, h, \text{aux}, M_0, M_1$ ; call this event  $K$ . Then the statistical distance of the distribution of  $[u_1^*]_4, [u_2^*]_4$  and  $[u_1^*]_5, [u_2^*]_5$  is less than or equal to  $2^{-s(\lambda)+1}$ . Define the event  $K_j$  for  $j \in \{3, 4\}$  as  $\langle [u_1^*]_j, [u_2^*]_j \rangle = \langle \alpha, \alpha' \rangle$  for some  $\alpha, \alpha'$  such that  $\log_{g_1} \alpha \neq \log_{g_2} \alpha'$ , and consider the conditioned games  $G_3[K \wedge K_3]$  and  $G_4[K \wedge K_4]$ . One can verify that these games are equivalent. It then follows from the second game playing lemma that  $|\text{Prob}[T_3] - \text{Prob}[T_4]| \leq 2^{-s(\lambda)+1}$ .

**Modification 5.** Finally, we modify game  $G_4$  to obtain game  $G_5$  by altering Step 6.

6. $v^* \leftarrow g_1^{r^3}$	$b \stackrel{r}{\leftarrow} \{0, 1\}, r_3 \stackrel{r}{\leftarrow} [q]$
-------------------------------	---

Condition on a fixed choice of all variables leading up to Step 6; that is, take  $K$  to be an event that fixes  $p, q, g_1, g_2, h, \text{aux}, M_0, M_1, u_1^*, u_2^*$ . Note that fixing the variables  $u_1^*, u_2^*$  implies that the random coins  $r, r'$  must also become fixed. This is not necessarily the case for all variables. For example,  $z_1, z_2$  are not fixed since conditioning on a fixed value of  $h$  does not force  $z_1$  or  $z_2$  to become entirely fixed.

Take any  $\alpha \in \langle g_1 \rangle$ . The probability distribution of  $[v^*]_5$  is clearly uniform over  $\langle g_1 \rangle$  when conditioned on  $K$ ; therefore the event  $[v^*]_5 = \alpha$  will have probability  $1/q$ , independent of  $b$ . Now the event  $[v^*]_4 = \alpha$  suggests that two equations must be satisfied as is modeled by the following system. The first equation is due to the conditioning on  $K$ .

$$\begin{bmatrix} 1 & w \\ r & wr' \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} \log_{g_1} h \\ \log_{g_1}(\alpha/M_b) \end{bmatrix} \quad (3)$$

The determinant of the system is  $w(r - r')$  which, based on based on our conditioning, must be nonzero. In other words,  $r = \log_{g_1} u_1^*$  and  $r' = \log_{g_2} u_2^*$  and  $r \neq r'$ . The probability that  $[v^*]_4 = \alpha$  must be  $1/q$  there are exactly  $q$  choices for  $z_1$  and  $z_2$  under the conditioning of  $K$ , and precisely one of them must be a solution to the above system. This is also independent of  $b$ .

Next we consider the equivalence of the two conditioned games  $G_4$  and  $G_5$  based on the events  $K \wedge ([v^*]_4 = \alpha)$  and  $K \wedge ([v^*]_5 = \alpha)$  respectively. These games are now quite different. In the conditioned game  $G_5$ , the variables  $z_1, z_2$  are subject only to the first equation in (??) and  $b$  is a random bit. In the conditioned game  $G_4$ , the distribution of  $z_1, z_2, b$  has exactly two values determined by both equations in (??). One choice is for  $b = 0$ , and the other for  $b = 1$ . Nevertheless, neither conditioned game employs  $z_1, z_2$  through  $\bar{z} = z_1 + wz_2$  in any step before Step 6, and of course,  $\bar{z}$  is identical in both conditionings (equal to  $\log_{g_1} h$ ). Moreover, the distribution of  $b$  is identical in both conditioned games; specifically, it is uniform over  $\{0, 1\}$ . From these two facts, we are able to derive that the two conditioned games are equivalent. As a result, we can apply the second game playing lemma to obtain  $\text{Prob}[T_4] = \text{Prob}[T_5]$ .

**Closing argument.** Through the previous series of modifications, it is now clear that  $\text{Prob}[T_5] = 1/2$  ( $b$  is never used before Step 8). To conclude the proof, note that the sequence of games reveals

$$\left| \text{Prob}[T_0] - \frac{1}{2} \right| \leq \text{Adv}_{\text{DDH}}^{\text{GGen}}(\lambda) + 2^{-s(\lambda)+3}.$$

Since both  $2^{-s(\lambda)}$  and  $\text{Adv}_{\text{DDH}}^{\text{GGen}}(\lambda)$  are assumed to be negligible in  $\lambda$ , this is also negligible in  $\lambda$ . This proves the following theorem:

**Theorem 11.1.1.** *The two-generator ElGamal public-key cryptosystem satisfies IND-CPA security under the Decisional Diffie-Hellman assumption.*

## 11.2 Step 2: The IND-CCA1 Version, “Lunch-Time Attacks”

As before, we will use  $\text{GGen}$  as the group generator that takes coin tosses  $\rho$  and length  $1^\lambda$  to produce  $p, q$  and  $g_1$ , where  $p$  is a  $\lambda$ -bit prime,  $q$  is an  $s(\lambda)$ -bit prime for a predetermined function  $s: \mathbb{N} \rightarrow \mathbb{N}$ ,

and  $g_1$  is an order  $q$  element in  $\mathbb{Z}_p^*$ . Note that we assume that  $s$  is selected so that  $2^{-s(\lambda)}$  is negligible in  $\lambda$ . We will employ the Decisional Diffie-Hellman assumption as defined in Section ??.

### 11.2.1 The CCA1-CS Public-Key Cryptosystem

We now present a CCA1-variant of the Cramer-Shoup public-key encryption scheme.

**Key-generation.** For input  $1^\lambda$ , select a  $\lambda$ -bit prime  $p$  such that  $g_1$  is a prime order  $q$  element for a  $q$  dividing  $p - 1$ , and  $q$  is of  $s(\lambda) < \lambda$  bits. Four random values are selected,  $w, z, x_1, x_2 \stackrel{\mathcal{R}}{\leftarrow} [q]$ . The public key is  $\langle p, q, g_1, g_2, h, c \rangle$  where  $g_1 \leftarrow g_1^w, h \leftarrow g_1^{z_1} g_2^{z_2}, c \leftarrow g_1^{x_1} g_2^{x_2}$ . The secret key is  $z_1, z_2, x_1, x_2$ .

**Encryption.** When given  $M \in \langle g_1 \rangle$ , the encryption function samples  $r \stackrel{\mathcal{R}}{\leftarrow} [q]$  and returns  $\langle u_1, u_2, v, e \rangle = \langle g_1^r, g_2^r, h^r M, c^r \rangle$ .

**Decryption.** The decryption function is given a ciphertext  $\langle u_1, u_2, v, e \rangle$  and tests if  $e = u_1^{x_1} u_2^{x_2}$ . If the test passes, the function returns  $v/u_1^{z_1} u_2^{z_2} \bmod p$ , otherwise it returns  $\perp$ .

#### The IND-CCA1 Game

The adversarial goal, CCA1 stands for the chosen-ciphertext attack with one round of decryption queries. An IND-CCA1 attack gets its nickname, the *the lunch-time attack* for allowing the scenario where an internal employee, with knowledge of a system, can break into someones computer over their lunch break.

Game $G_0$ on Input $1^\lambda$	Random Variables
1. $\langle p, q, g_1 \rangle \leftarrow \text{GGen}(1^\lambda; \rho)$	$\rho \leftarrow \text{Coins}$
2. $g_2 \leftarrow g_1^w$	$w \stackrel{\mathcal{R}}{\leftarrow} [q]$
3. $c \leftarrow g_1^{x_1} g_2^{x_2}$	
4. $h \leftarrow g_1^{z_1} g_2^{z_2}$	$z_1, z_2 \stackrel{\mathcal{R}}{\leftarrow} [q]$
5. $\langle \text{aux}, M_0, M_1 \rangle \leftarrow \mathcal{A}^{\text{Dec}(\cdot)[Q]}(\text{stage}_1, g_1, g_2, c, h; \rho_1)$	$\rho_1 \stackrel{\mathcal{R}}{\leftarrow} \text{Coins}_1$
6. $u_1^* \leftarrow g_1^r; u_2^* \leftarrow g_2^r$	$r \stackrel{\mathcal{R}}{\leftarrow} [q]$
7. $v^* \leftarrow h^r M_b; e^* \leftarrow c^r$	$b \stackrel{\mathcal{R}}{\leftarrow} \{0, 1\}$
8. $b^* \leftarrow \mathcal{A}(\text{stage}_2, u_1^*, u_2^*, v^*, e^*; \rho_2)$	$\rho_2 \stackrel{\mathcal{R}}{\leftarrow} \text{Coins}_2$
9. if $b = b^*$ return 1 else return 0	

During Step 5, the decryption oracle  $\text{Dec}(\cdot)$  is queried  $Q$  times. When given a ciphertext  $\langle u_1, u_2, v, e \rangle$ , it tests if  $e = u_1^{x_1} u_2^{x_2}$ . If and when the test passes, the decryption oracle returns  $v/u_1^{z_1} u_2^{z_2}$ ; otherwise  $\perp$  is returned.

#### Proof of Security

We present the proof of security structured as a sequence of games.

**Modification 1.** We perform the following modification to game  $G_0$  to obtain game  $G_1$ :

7. $v^* \leftarrow (u_1^*)^{z_1} (u_2^*)^{z_2} M_b; e^* \leftarrow (u_1^*)^{x_1} (u_2^*)^{x_2}$	$b \stackrel{\mathcal{R}}{\leftarrow} \{0, 1\}$
---	---

This is again a syntactic modification since  $(u_1^*)^{z_1} (u_2^*)^{z_2} M_b = h^r M_b$  and  $(u_1^*)^{x_1} (u_2^*)^{x_2} = c^r$ . It directly follows that  $\text{Prob}[T_0] = \text{Prob}[T_1]$ .

**Modification 2.** Modify Step 6 of  $G_1$  to obtain game  $G_2$ .

6. $u_1^* \leftarrow g_1^r; u_2^* \leftarrow g_2^{r'}$	$r, r' \stackrel{\mathcal{R}}{\leftarrow} [q]$
--	--

Note that the variables  $w, r, r'$  are not explicitly used anywhere in either game  $G_1$  or  $G_2$ . Moreover, for any group description  $\langle p, q, g_1 \rangle$  produced by  $\text{GGen}$ , the triple  $\langle g_2, u_1^*, u_2^* \rangle$  is distributed as

a DDH triple in game  $G_1$ , and as a random triple drawn from  $\langle g_1 \rangle$  in game  $G_2$ . By the third game-playing lemma,  $|\text{Prob}[T_1] - \text{Prob}[T_2]| \leq \text{Adv}_{\text{DDH}}^{\text{Gen}}(\lambda)$ .

**Modification 3.** Modify game  $G_2$  to obtain game  $G_3$  by

2. $g_2 \leftarrow g_1^w$	$w \xleftarrow{\mathcal{R}} [q-1]$
---------------------------	------------------------------------

Conditioned on any choice of  $p, q, g_1$ , the statistical distance between the probability distribution of  $[g_2]_2$  and  $[g_2]_3$  is  $1/q$ . Indeed,

$$\frac{1}{2} \left[ (q-1) \left| \frac{1}{q} - \frac{1}{q-1} \right| + \frac{1}{q} \right] = \frac{1}{q} \leq \frac{1}{2^{s(\lambda)-1}}.$$

The variable  $w$  is used only through  $g_2$  in games  $G_2, G_3$ , so one can verify that conditioned on a choice of  $p, q, g_1$ , and  $[g_2]_2 = \alpha$ , the conditioned game  $G_2$  is equivalent to game  $G_3$  when conditioned on the same choice of  $p, q, g_1$ , and  $[g_2]_3 = \alpha$ . From the second game-playing lemma then,  $|\text{Prob}[T_2] - \text{Prob}[T_3]| \leq 2^{-s(\lambda)+1}$ .

**Modification [4,  $i$ ]** where  $i = 1, \dots, Q$ . We modify the answer to the  $i$ th query to the decryption oracle in Step 5 so that it operates as follows:

Set the variables

$$\bar{z} \leftarrow z_1 + wz_2; \bar{x} \leftarrow x_1 + wx_2$$

Given a ciphertext  $\langle u_1, u_2, v, e \rangle$ , test if  $u_2 = u_1^w$  and  $e = u_1^{\bar{x}}$ . If both tests pass, return the value  $v/u_1^{\bar{z}}$ ; otherwise return  $\perp$ .

Games  $G_{4,i}$  and  $G_{4,i-1}$  are clearly defined over an identical probability space (we assume  $G_{4,0} = G_3$ ). Consider the event  $F$  over this joint probability space that the adversary produces a ciphertext as its  $i$ th query that the two oracles answer differently. If a ciphertext  $\langle u_1, u_2, v, e \rangle$  passes the test of game  $G_{4,i}$ , the answer must be the same; specifically,  $u_2 = u_1^w$  implies  $e = u_1^{\bar{x}} = u_1^{x_1} u_2^{x_2}$ , so the ciphertext also passes the test of game  $G_{4,i-1}$ . The answer of game  $G_{4,i}$  is then  $v/u_1^{\bar{z}}$ , which equals  $v/u_1^{z_1} u_2^{z_2}$ . If the ciphertext fails both tests, then the answers are again identical.

The only interesting case is then when the  $i$ th query passes the test performed in game  $G_{4,i-1}$ , but fails the test of game  $G_{4,i}$ . This implies that  $e = u_1^{x_1} u_2^{x_2}$ , while either  $u_2 \neq u_1^w$  or  $e \neq u_1^{\bar{x}}$ . Note that if  $u_2 = u_1^w$ , it would mandate that  $e = u_1^{\bar{x}}$ . We can therefore conclude that if  $F$  occurs,  $u_2 \neq u_1^w$  and the corresponding ciphertext passes the test of game  $G_{4,i-1}$ .

Notice that whenever the event  $\neg F$  occurs, the two games proceed identically: when conditioning on the event  $\neg F$ , the two games are identical. It follows from the first-game playing lemma that  $|\text{Prob}[T_{4,i-1}] - \text{Prob}[T_{4,i}]| \leq \text{Prob}[F]$ . We have now found a bound on  $\text{Prob}[F]$ . Let us condition on a fixed choice of  $p, q, g_1, g_2, c$  and suppose  $F$  happens. Then in the  $i$ th query,  $\mathcal{A}$  outputs  $u_1, u_2, e$  with  $\log_{g_1} u_1 \neq \log_{g_2} u_2$  and  $e = u_1^{x_1} u_2^{x_2}$ . Note that for the first  $i-1$  queries, the random coin tosses  $x_1, x_2$  in the view of  $\mathcal{A}$  must satisfy the first equation in the system

$$\begin{bmatrix} 1 & w \\ \log_{g_1} u_1 & w \log_{g_2} u_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \bar{x} \\ \log_{g_1} e \end{bmatrix}.$$

This follows from the fact that all queries made by the adversary prior to posing its  $i$ th query are answered by employing  $\bar{x}$  instead of  $x_1, x_2$ . This system has a full-rank minor with nonzero determinant  $w(\log_{g_2} u_2 - \log_{g_1} u_1)$ ; thus, the event  $F$  occurs with probability  $1/q \leq 2^{-s(\lambda)+1}$ , conditioned on the choice of  $p, q, g_1, g_2, c$ . Then  $\text{Prob}[F] \leq 2^{-s(\lambda)+1}$ .

**Modification 5.** We now modify game  $G_4 = G_{4,Q}$  to obtain game  $G_5$ :

6. $u_1^* \leftarrow g_1^r; u_2^* \leftarrow g_2^{r'}$	$\langle r, r' \rangle \xleftarrow{\mathcal{R}} ([q] \times [q])^\neq$
--	--

It is easy to see that conditioning on  $p, q, g_1, g_2, c, h, \mathbf{aux}, M_0, M_1$  the statistical distance of the distribution of  $[u_1^*]_4, [u_2^*]_4$  and  $[u_1^*]_5, [u_2^*]_5$  is less than or equal to  $2^{-s(\lambda)+1}$ .

Let us now condition on some choice of  $p, q, g_1, g_2, c, h, \mathbf{aux}, M_0, M_1$  and call this the event  $K$ . Consider the event  $K_j$  for  $j \in \{4, 5\}$ , defined as  $\langle [u_1^*]_j, [u_2^*]_j \rangle = \langle \alpha, \alpha' \rangle$  for some  $\alpha, \alpha'$  with  $\log_{g_1} \alpha \neq \log_{g_2} \alpha'$ . We consider now the conditioned games  $G_4[K \wedge K_4]$  and  $G_5[K \wedge K_5]$ . It is easy to verify that the two conditioned games are equivalent. It follows from the second game playing lemma that  $|\text{Prob}[T_{4,Q}] - \text{Prob}[T_5]| \leq 2^{-s(\lambda)+1}$ .

**Modification 6.** Lastly, we modify Step 7 of game  $G_5$  to obtain game  $G_6$ .

$7. \quad v^* \leftarrow g_1^{r''}; \quad e^* \leftarrow (u_1^*)^{x_1} (u_2^*)^{x_2}$	$b \stackrel{r}{\leftarrow} \{0, 1\}, \quad r'' \stackrel{r}{\leftarrow} [q]$
---	---

Consider an event  $K$  that fixes  $p, q, g_1, g_2, h, \mathbf{aux}, M_0, M_1, u_1^*, u_2^*$ . In fixing the variables  $u_1^*, u_2^*$ , the random coins  $r, r'$  must also become fixed. This is not the case for  $z_1, z_2$  since conditioning on a fixed value of  $h$  does not force  $z_1, z_2$  to become entirely fixed.

Now take an  $\alpha \in \langle g_1 \rangle$ . The probability distribution of  $[v^*]_6$  is clearly uniform over  $\langle g_1 \rangle$  conditioned on  $K$ ; therefore, the event  $[v^*]_6 = \alpha$  has probability  $1/q$ , which is independent of  $b$ . In looking at the event  $[v^*]_5 = \alpha$ , we see that the following system must be satisfied where the first equation is due to the conditioning on  $K$ :

$$\begin{bmatrix} 1 & w \\ r & wr' \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} \log_{g_1} h \\ \log_{g_1} (\alpha/M_b) \end{bmatrix}. \quad (4)$$

The determinant of the system is  $w(r' - r)$  and, based on our conditioning, must be nonzero ( $r = \log_{g_1} u_1^*$  and  $r' = \log_{g_2} u_2^*$ ; moreover,  $r \neq r'$ ). It then follows that the probability of the event  $[v^*]_5 = \alpha$  must be  $1/q$  since there are exactly  $q$  choices for selecting  $z_1, z_2$  under the conditioning of  $K$  and precisely one of them will be the solution to (??). This is also independent of the choice of  $b$ .

Next we have to consider the equivalence of the two conditioned games  $G_5$  and  $G_6$  based on the events  $K \wedge ([v^*]_5 = \alpha)$  and  $K \wedge ([v^*]_6 = \alpha)$  respectively. The two conditioned games are now quite different. Indeed, in the conditioned game  $G_6$ , the variables  $z_1, z_2$  are subject only to the first equation in (??) and  $b$  is left as a random bit. In the conditioned game  $G_5$ , the random variables  $z_1, z_2$ , and  $b$  do not follow the same distribution. Here the variables are subject to both equations in (??), so their distribution has exactly two possible values, one for  $b = 0$  and one for  $b = 1$ . Nevertheless, both games only employ  $z_1, z_2$  through  $\bar{z} = z_1 + wz_2$  after Step 7, and of course,  $\bar{z}$  is identical in both conditionings (equal to  $\log_{g_1} h$ ). The distribution of  $b$  is also identical in both conditioned games; it is uniform over  $\{0, 1\}$ . Together, these facts imply that the two conditioned games are equivalent. As a result, we can apply the second game playing lemma to obtain  $\text{Prob}[T_5] = \text{Prob}[T_6]$ .

**Closing argument.** It is now clear that  $\text{Prob}[T_6] = 1/2$  since  $b$  is not used before Step 9. In conclusion,

$$\left| \text{Prob}[T_0] - \frac{1}{2} \right| \leq \text{Adv}_{\text{DDH}}^{\text{GGen}}(\lambda) + (Q + 2)2^{-s(\lambda)+1}.$$

This is negligible in  $\lambda$  since  $Q$  is polynomial bounded in  $\lambda$  and  $\text{Adv}_{\text{DDH}}^{\text{GGen}}(\lambda)$  is assumed to be negligible in  $\lambda$ . This proves the following theorem:

**Theorem 11.2.1.** *The CCA1-CS public-key cryptosystem satisfies IND-CCA1 (lunch-time attack) security under the decisional Diffie-Hellman assumption.*

### 11.3 Step 3: The IND-CCA2 Version

Let  $\text{GGen}$  be a group generator that takes coin tosses  $\rho$  and length  $1^\lambda$  and produces  $p, q$  and  $g_1$  such that  $p$  is a  $\lambda$ -bit prime,  $q$  is an  $s(\lambda)$ -bit prime for a predetermined function  $s: \mathbb{N} \rightarrow \mathbb{N}$ , and  $g_1$  is an order  $q$  element from  $\mathbb{Z}_p^*$ . We will assume that  $s$  is selected so that  $2^{-s(\lambda)}$  is negligible in  $\lambda$ .



**Assumption 1.** Our first assumption is the decisional Diffie-Hellman assumption as defined in Section ??.

**Assumption 2.** There is a family  $\text{UOWHF}(p, q, g_1)$  of functions  $\langle g_1 \rangle \times \langle g_1 \rangle \times \langle g_1 \rangle \rightarrow \mathbb{Z}_q$ , where  $\langle p, q, g_1 \rangle$  is drawn from  $\text{GGen}(1^\lambda)$  so that the function

$$\text{Adv}_{\text{UOWHF}}^{\text{GGen}}(\lambda) = \max_{\mathcal{A}} \{ \text{Prob}[\mathcal{A}(x, \text{desc}\mathcal{H}) = x'] \wedge (x \neq x') \wedge (\mathcal{H}(x) = \mathcal{H}(x')) \}$$

is negligible in  $\lambda$ . The maximum is taken over all PPT  $\mathcal{A}$  and the probability is taken over all choices of  $\langle p, q, g_1 \rangle$  from  $\text{GGen}(1^\lambda)$ ,  $\mathcal{H}$  from  $\text{UOWHF}(p, q, g_1)$ , and  $x$  from  $\langle g_1 \rangle \times \langle g_1 \rangle \times \langle g_1 \rangle$ . The string  $\text{desc}\mathcal{H}$  is a circuit describing the function  $\mathcal{H}$ .

### 11.3.1 The CS Public-Key Cryptosystem

The Cramer-Shoup (CS) public-key encryption scheme is as follows:

**Key-generation.** For input  $1^\lambda$ , select a  $\lambda$ -bit prime  $p$  such that  $g_1$  is a prime order  $q$  element,  $q$  divides  $p - 1$ , and  $q$  is of  $s(\lambda) < \lambda$  bits. Select six random values,  $w, z, x_1, x_2, y_1, y_2 \xleftarrow{\mathcal{R}} [q]$ . Sample a universal one-way hash function  $\mathcal{H}$  from  $\text{UOWHF}(1^\lambda)$ :  $\langle g_1 \rangle \times \langle g_1 \rangle \times \langle g_1 \rangle \rightarrow \mathbb{Z}_q$ . The public key is  $\langle p, q, g_1, g_2, h, c, d, \text{desc}\mathcal{H} \rangle$  where  $g_2 \leftarrow g_1^w, h \leftarrow g_1^z, c \leftarrow g_1^{x_1} g_2^{x_2}, d \leftarrow g_1^{y_1} g_2^{y_2}$ . The secret key is  $z_1, z_2, x_1, x_2, y_1, y_2$ .

**Encryption.** The encryption function takes a value  $M \in \langle g_1 \rangle$ , samples  $r \xleftarrow{\mathcal{R}} [q]$ , and returns  $\langle u_1, u_2, v, e \rangle = \langle g_1^r, g_2^r, h^r M, c^r d^{hr} \rangle$  where  $h \leftarrow \mathcal{H}(u_1, u_2, v)$ .

**Decryption.** When given a ciphertext  $\langle u_1, u_2, v, e \rangle$ , the decryption function computes  $h \leftarrow \mathcal{H}(u_1, u_2, v)$  and tests if  $e = u_1^{x_1 + y_2 h} u_2^{x_2 + y_1 h}$ . If the test passes, it returns  $v / u_1^{z_1} u_2^{z_2} \bmod p$ ; otherwise it returns  $\perp$ .

**The IND-CCA2 Game.** IND-CCA2 now allows an adversary to make two rounds of decryption queries in a chosen-ciphertext attack.

Game $G_0$ on Input $1^\lambda$	Random variables
1. $\langle p, q, g_1 \rangle \leftarrow \text{GGen}(1^\lambda; \rho)$	$\rho \leftarrow \text{Coins}$
2. $\mathcal{H} \leftarrow \text{UOWHF}(1^\lambda, \rho')$	$\rho' \leftarrow \text{Coins}'$
3. $g_2 \leftarrow g_1^w$	$w \xleftarrow{\mathcal{R}} [q]$
4. $c \leftarrow g_1^{x_1} g_2^{x_2}; d \leftarrow g_1^{y_1} g_2^{y_2}$	$x_1, x_2, y_1, y_2 \xleftarrow{\mathcal{R}} [q]$
5. $h \leftarrow g_1^{z_1} g_2^{z_2}$	$z_1, z_2 \xleftarrow{\mathcal{R}} [q]$
6. $\langle \text{aux}, M_0, M_1 \rangle \leftarrow \mathcal{A}^{\text{Dec}(\cdot)[Q_1]}(\text{stage}_1, g_1, g_2, c, d, h; \rho_1)$	$\rho_1 \xleftarrow{\mathcal{R}} \text{Coins}_1$
7. $u_1^* \leftarrow g_1^r; u_2^* \leftarrow g_2^r$	$r \xleftarrow{\mathcal{R}} [q]$
8. $v^* \leftarrow h^r M_b; e^* \leftarrow c^r d^{r\mathcal{H}(u_1^*, u_2^*, v^*)}$	$b \xleftarrow{\mathcal{R}} \{0, 1\}$
9. $b^* \leftarrow \mathcal{A}^{\text{Dec}'(\cdot)[Q_2]}(\text{stage}_2, u_1^*, u_2^*, v^*, e^*; \rho_2)$	$\rho_2 \xleftarrow{\mathcal{R}} \text{Coins}_2$
10. if $b = b^*$ return 1 else return 0	

The decryption oracle  $\text{Dec}(\cdot)$  is queried  $Q_1$  times during Step 6 and  $\text{Dec}'(\cdot)$  is queried  $Q_2$  times in Step 9. The operation of  $\text{Dec}(\cdot)$  is defined below.

Given a ciphertext  $\langle u_1, u_2, v, e \rangle$ , test if  $e = u_1^{x_1 + y_2 h} u_2^{x_2 + y_1 h}$  where  $h = \mathcal{H}(u_1, u_2, v)$ . If the test passes, return the value  $v / u_1^{z_1} u_2^{z_2}$ ; otherwise return  $\perp$ .

$\text{Dec}'$  is similarly defined with the restriction that the challenge ciphertext  $\langle u_1^*, u_2^*, v^*, e^* \rangle$  results immediately in  $\perp$ .

### Proof of Security

We present the proof of security structured as a sequence of games.

**Modification 1.** Alter Step 8 of game  $G_0$  to obtain game  $G_1$ .

8. $v^* \leftarrow (u_1^*)^{z_1} (u_2^*)^{z_2} M_b$ ; $e^* \leftarrow (u_1^*)^{x_1 + y_1 h} (u_2^*)^{x_2 + y_2 h}$ ; $h \leftarrow \mathcal{H}(u_1^*, u_2^*, v^*)$	$b \xleftarrow{\mathcal{R}} \{0, 1\}$
---	---------------------------------------

As with the first modification in the previous two security proofs, this is used to provide a useful syntactic adjustment without altering the probability distributions. This follows from the fact that  $(u_1^*)^{z_1} (u_2^*)^{z_2} M_b = h^r M_b$ ,  $(u_1^*)^{x_1} (u_2^*)^{x_2} = c^r$ , and  $(u_1^*)^{y_1} (u_2^*)^{y_2} = d^r$ . Therefore,  $\text{Prob}[T_0] = \text{Prob}[T_1]$ .

**Modification 2.** Perform the following modification on game  $G_1$  to obtain game  $G_2$ :

7. $u_1^* \leftarrow g_1^r$ ; $u_2^* \leftarrow g_2^{r'}$	$r, r' \xleftarrow{\mathcal{R}} [q]$
---	--------------------------------------

Observe that the variables  $w, r, r'$  are not explicitly used anywhere in  $G_1$  or  $G_2$ . Moreover, for any group description  $\langle p, q, g_1 \rangle$  that  $\text{GGen}$  produces, the triple  $\langle g_2, u_1^*, u_2^* \rangle$  is distributed as a DDH triple in game  $G_1$  and as a random triple drawn from  $\langle g_1 \rangle$  in game  $G_2$ . It follows from the third game-playing lemma that  $|\text{Prob}[T_1] - \text{Prob}[T_2]| \leq \text{Adv}_{\text{DDH}}^{\text{GGen}}(\lambda)$ .

**Modification 3.** Modify game  $G_2$  as follows to obtain game  $G_3$ :

2. $g_2 \leftarrow g_1^w$	$w \xleftarrow{\mathcal{R}} [q-1]$
---------------------------	------------------------------------

Conditioned on any choice of  $p, q, g_1, \mathcal{H}$ , the statistical distance between the probability distribution of  $[g_2]_2$  and  $[g_2]_3$  is  $1/q$ ; indeed, we have

$$\frac{1}{2} \left[ (q-1) \left| \frac{1}{q} - \frac{1}{q-1} \right| + \frac{1}{q} \right] = \frac{1}{q} \leq \frac{1}{2^{s(\lambda)-1}}.$$

The variable  $w$  is only used through  $g_2$  in  $G_2$  and  $G_3$ , so it is easy to verify that when conditioned on any choice of  $p, q, g_1, \mathcal{H}$  and  $[g_2]_2 = \alpha$ , the conditioned game  $G_2$  is equivalent to the conditioned game  $G_3$  on the same  $p, q, g_1, \mathcal{H}$  and  $[g_2]_3 = \alpha$ . We conclude from the second game-playing lemma that  $|\text{Prob}[T_2] - \text{Prob}[T_3]| \leq 2^{-s(\lambda)+1}$ .

**Modification [4,  $i$ ]** where  $i = 1, \dots, Q_1$ . Modify the answer to the  $i$ th query to the decryption oracle in Step 6.

Set the variables

$$\bar{z} \leftarrow z_1 + wz_2; \bar{x} \leftarrow x_1 + wx_2; \bar{y} \leftarrow y_1 + wy_2$$

Given a ciphertext  $\langle u_1, u_2, v, e \rangle$ , test if  $u_2 = u_1^w$  and  $e = u_1^{\bar{x} + \bar{y}h}$ , where  $h = \mathcal{H}(u_1, u_2, v)$ .

If both tests pass, return the value  $v/u_1^{\bar{z}}$ ; otherwise return  $\perp$ .

Clearly  $G_{4,i}$  and  $G_{4,i-1}$  are defined over an identical probability space (we assume  $G_{4,0} = G_3$ ). Consider the event  $F$  over this joint probability space that the adversary produces a ciphertext as its  $i$ th query so that the answer of the two oracles is different on this ciphertext.

If a ciphertext  $\langle u_1, u_2, v, e \rangle$  passes the test of game  $G_{4,i}$ , the answer must be the same in  $G_{4,i-1}$ . Specifically, we have that  $u_2 = u_1^w$ , so  $e = u_1^{\bar{x} + \bar{y}h} = u_1^{x_1 + y_1 h} u_2^{x_2 + y_2 h}$ ; that is, the ciphertext also passes the test of game  $G_{4,i-1}$ . The answer of game  $G_{4,i}$  equals  $v/u_1^{\bar{z}}$ , which is equal to  $v/u_1^{z_1} u_2^{z_2}$ . On the other hand, if the ciphertext fails both tests, the answers are identical in  $G_{4,i}$  and  $G_{4,i-1}$ .

The only interesting case is then when the  $i$ th query passes the test performed in game  $G_{4,i-1}$ , but fails the test of game  $G_{4,i}$ . Recall that in this case  $e = u_1^{x_1 + y_1 h} u_2^{x_2 + y_2 h}$ , while either  $u_2 \neq u_1^w$  or  $e \neq u_1^{\bar{x} + \bar{y}h}$ . Note that we cannot have  $u_2 = u_1^w$  since this implies  $e = u_1^{\bar{x} + \bar{y}h}$ . If the event  $F$  occurs, we can assume  $u_2 \neq u_1^w$  and the corresponding ciphertext passes the test of game  $G_{4,i-1}$ .

Whenever the event  $\neg F$  happens, the two games proceed identically (i.e., when conditioning on the event  $\neg F$ , the two games are identical). It follows from the first-game playing lemma that

$$|\text{Prob}[T_{4,i-1}] - \text{Prob}[T_{4,i}]| \leq \text{Prob}[F].$$

Thus we have found a bound on  $\text{Prob}[F]$ . Fix a choice of  $p, q, g_1, g_2, c, d$  and suppose  $F$  occurs. Then in the  $i$ th query,  $\mathcal{A}$  outputs  $u_1, u_2, e$  with  $\log_{g_1} u_1 \neq \log_{g_2} u_2$  and  $e = u_1^{x_1+y_1h} u_2^{x_2+y_2h}$ . Observe now that the random coin tosses  $x_1, x_2, y_1, y_2$ , in the view of  $\mathcal{A}$  up until the  $i$ th query, must satisfy the first two equations of

$$\begin{bmatrix} 1 & w & 0 & 0 \\ 0 & 0 & 1 & w \\ \log_{g_1} u_1 & w \log_{g_2} u_2 & h \log_{g_1} u_1 & wh \log_{g_2} u_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} \bar{x} \\ \bar{y} \\ \log_{g_1} e \end{bmatrix} \quad (5)$$

This follows from the fact that all queries made by the adversary prior to posing its  $i$ th query are answered using  $\bar{x}$  and  $\bar{y}$ , not  $x_1, x_2, y_1, y_2$ . The  $i$ th query itself forms the third equation of (??).

This system has a full-rank minor with nonzero determinant  $w(\log_{g_1} u_1 - \log_{g_2} u_2)$ ; thus, the event  $F$  occurs with probability  $1/q \leq 2^{-s(\lambda)+1}$  conditioned on the choice of  $p, q, g_1, g_2, c, d$ . It follows that  $\text{Prob}[F] \leq 2^{-s(\lambda)+1}$ .

**Modification 5.** Continuing with our notation from Modification 4 in Section ??, modify  $G_4 = G_{4,Q_2}$  to obtain game  $G_5$ .

7. $u_1^* \leftarrow g_1^r; u_2^* \leftarrow g_2^{r'}$	$\langle r, r' \rangle \xleftarrow{\mathcal{R}} ([q] \times [q])^\neq$
--	--

If we condition on  $p, q, g_1, \mathcal{H}, g_2, c, d, h, \text{aux}, M_0, M_1$ , the statistical distance of the distribution of  $[u_1^*]_4, [u_2^*]_4$  and  $[u_1^*]_5, [u_2^*]_5$  is less than or equal to  $2^{-s(\lambda)+1}$ . Suppose we condition on some choice of  $p, q, g_1, \mathcal{H}, g_2, c, d, h, \text{aux}, M_0, M_1$ ; call this the event  $K$  and define  $K_j$  for  $j \in \{4, 5\}$  as the event  $\langle [u_1^*]_j, [u_2^*]_j \rangle = \langle \alpha, \alpha' \rangle$  for some  $\alpha, \alpha'$  with  $\log_{g_1} \alpha \neq \log_{g_2} \alpha'$ . Then the conditioned games  $G_4[K \wedge K_4]$  and  $G_5[K \wedge K_5]$  are equivalent. By the second game playing lemma,  $|\text{Prob}[T_4] - \text{Prob}[T_5]| \leq 2^{-s(\lambda)+1}$ .

**Modification 6.** Modify the operation of the decryption oracle  $\text{Dec}'(\cdot)$  in Step 9 to form  $G_6$ .

Set the variables

$$\bar{z} \leftarrow z_1 + wz_2; \bar{x} \leftarrow x_1 + wx_2; \bar{y} \leftarrow y_1 + wy_2.$$

Given a ciphertext  $\langle u_1, u_2, v, e \rangle$ , test if  $u_2 = u_1^w$  and  $e = u_1^{\bar{x} + \bar{y}h}$ , where  $h = \mathcal{H}(u_1, u_2, v)$ .

If both tests pass, return  $v/u_1^{\bar{z}}$ ; otherwise return  $\perp$ .

Define  $F$  to be the event that the oracle responds different in some query of the second stage of the adversary. Games  $G_5$  and  $G_6$  are defined over the same probability space. Since  $G_5$  and  $G_6$  proceed identically as long as  $\neg F$  occurs,  $|\text{Prob}[T_6] - \text{Prob}[T_5]| \leq \text{Prob}[F]$  by the first game playing lemma.

If  $F_6$  occurs, the adversary produces a ciphertext  $\langle u_1, u_2, v, e \rangle$  in his second stage such that  $e = u_1^{x_1+y_1h} u_2^{x_2+y_2h}$ , but  $\log_{g_1} u_1 \neq \log_{g_2} u_2$  by the same reasoning as in  $G_{4,i}$ . Define  $\text{Coll}$  to be the event that the adversary produces a ciphertext  $\langle u_1, u_2, v, e \rangle$  during his second stage so that  $\langle u_1, u_2, v, e \rangle \neq \langle u_1^*, u_2^*, v^*, e^* \rangle$ , but  $\mathcal{H}(u_1, u_2, v) = \mathcal{H}(u_1^*, u_2^*, v^*)$  (i.e., a collision for  $\mathcal{H}$ ).

We will investigate the event  $\neg \text{Coll} \cap F$ . Note that  $F = F_1 \cup \dots \cup F_{Q_2}$ , where  $F_i$  denotes the event that the adversary produces a ciphertext in its  $i$ th query for which the condition of  $F$  holds. Using this, we will first place a bound on the probability  $\neg \text{Coll} \cap \text{Prob}[F_i]$ , and then apply the union bound to obtain an upper bound for the probability  $\neg \text{Coll} \cap F$ . Let us condition over  $p, q, g_1, \mathcal{H}, g_2, c, d, h, \text{aux}, M_0, M_1, u_1^*, u_2^*, v^*, e^*$ . Suppose that prior to the adversary making his  $i$ th

query in  $\text{stage}_2$ , the values  $x_1, x_2, y_1, y_2$  satisfy the first three of the equations of the following system in the view of the adversary.

$$\begin{bmatrix} 1 & w & 0 & 0 \\ 0 & 0 & 1 & w \\ r & wr' & h^*r & wh^*r' \\ \log_{g_1} u_1 & w \log_{g_2} u_2 & h \log_{g_1} u_1 & wh \log_{g_2} u_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} \bar{x} \\ \bar{y} \\ \log_{g_1} e^* \\ \log_{g_1} e \end{bmatrix}.$$

The adversary's  $i$ th query yields the fourth equation in the above system. The determinant is  $w^2(r - r')(h - h^*)(\log_{g_1} u_1 - \log_{g_2} u_2)$ , which is nonzero since we assume  $\neg\text{Coll}$ . Before his  $i$ th query in Step 9, the adversary cannot obtain any information regarding  $x_1, x_2, y_1, y_2$ , beyond what is contained in the first three equations; therefore  $\text{Prob}[F_i \cap \neg\text{Coll}] \leq 1/q$ . In general,  $\text{Prob}[F_i \cap \neg\text{Coll}] \leq 2^{-s(\lambda)+1}$ . It follows then that

$$\text{Prob}[\neg\text{Coll} \cap F] \leq Q_2 \cdot 2^{-s(\lambda)+1}.$$

Finally, using the fact that

$$\text{Prob}[F] = \text{Prob}[\neg\text{Coll} \cap F] + \text{Prob}[\text{Coll} \cap F] \leq \text{Prob}[\neg\text{Coll} \cap F] + \text{Prob}[\text{Coll}],$$

we obtain the result

$$|\text{Prob}[T_6] - \text{Prob}[T_5]| \leq Q_2 \cdot 2^{-s(\lambda)+1} + \text{Prob}[\text{Coll}].$$

We still need a bound on  $\text{Prob}[\text{Coll}]$ , which we will find in the next game.

**Modification 7.** Alter Step 8 of game  $G_6$  to obtain the game  $G_7$ .

8. $v^* \leftarrow g_1^{r^3}; e^* \leftarrow (u_1^*)^{x_1} (u_2^*)^{x_2}$	$b \stackrel{r}{\leftarrow} \{0, 1\}, r_3 \stackrel{r}{\leftarrow} [q]$
---	---

Let us condition on a fixed choice of all variables prior to Step 7: i.e., we consider an event  $K$  that fixes  $p, q, g_1, \mathcal{H}, g_2, h, \text{aux}, M_0, M_1, u_1^*, u_2^*$ . Note that fixing the variables  $u_1^*, u_2^*$  implies that the random coins  $r, r'$  also become fixed. Now consider some  $\alpha \in \langle g_1 \rangle$ . The probability distribution of  $[v^*]_7$  is clearly uniform over  $\langle g_1 \rangle$  conditioned on  $K$  and thus the event  $[v^*]_7 = \alpha$  will have probability  $1/q$ . The event  $[v^*]_6 = \alpha$  suggests that the following two equations must be satisfied; the first is due to the conditioning on  $K$ :

$$\begin{bmatrix} 1 & w \\ r & wr' \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} \log_{g_1} h \\ \log_{g_1} (\alpha/M_b) \end{bmatrix}$$

This system's determinant is  $w(r' - r)$ , and is therefore nonzero from our conditioning:  $r = \log_{g_1} u_1^*, r' = \log_{g_2} u_2^*$ , and  $r \neq r'$ . It follows that the probability of the event  $[v^*]_6 = \alpha$  must be  $1/q$  since there are exactly  $q$  choices for selecting  $z_1, z_2$ , and regardless of the choice of  $b$ , precisely one of these choices will be the solution to the system.

Next we consider the equivalence of the two conditioned games  $G_6$  and  $G_7$  based on the events  $K \wedge ([v^*]_6 = \alpha)$  and  $K \wedge ([v^*]_7 = \alpha)$  respectively. These conditioned games are now quite different. In the conditioned game  $G_6$ , the variables  $z_1, z_2$  are subject only to the first equation of the system and  $b$  is a random bit uniformly distributed over  $\{0, 1\}$ . In the conditioned game  $G_7$ , the random variables  $z_1, z_2$  are not random since they must assume with probability  $1/2$  one of the two values allowed by the above system (one is selected for  $b = 0$  and one is selected by  $b = 1$ ). Progressing through the steps in both games however, one can verify that neither game employs  $z_1, z_2$  other than through  $\bar{z} = z_1 + wz_2$  after Step 8. The variable  $\bar{z}$  is clearly identical in both conditionings (equal to  $\log_{g_1} h$ ) and the conditional probability distribution of  $b$  in  $G_6$  is still uniform over  $\{0, 1\}$ . The two games are therefore equivalent and by the second game playing lemma,  $\text{Prob}[T_6] = \text{Prob}[T_7]$ .

Now let  $\text{Coll}_6$  be the event  $\text{Coll}$  as defined in Modification 6:  $\text{Coll}_6$  is the event that the adversary produces a collision for  $\mathcal{H}$  in the second stage. Define a similar event  $\text{Coll}_7$  in game  $G_7$ .

Consider now a modified game  $G_6$ , called  $G'_6$  that operates exactly as  $G_6$  except in Step 10:

10. If $\mathcal{A}$ finds a collision $\langle u_1, u_2, v \rangle$ of $\mathcal{H}$ against $\langle u_1^*, u_2^*, v^* \rangle$ in Step 9 return 1 else 0	
---	--

We similarly define a modified game  $G'_7$ . It then holds for the winning probabilities of the two games, that  $\text{Prob}[T'_6] = \text{Prob}[\text{Coll}_6]$  and  $\text{Prob}[T'_7] = \text{Prob}[\text{Coll}_7]$ . By the same arguments used in the transition from  $G_6$  to  $G_7$ , we find  $\text{Prob}[\text{Coll}_6] = \text{Prob}[\text{Coll}_7]$ .

**Modification 8.** We perform our final modification by changing Step 7 of game  $G'_7$  to obtain  $G'_8$ .

7. $u_1^* \leftarrow g_1^r; u_2^* \leftarrow g_2^{r'}$	$r, r' \xleftarrow{r} [q]$
--	----------------------------

Here we are essentially reversing Modification 5. Note that  $r, r'$  are not explicitly used in either  $G'_7$  or  $G'_8$ , so a direct application of the second game-playing lemma proves

$$|\text{Prob}[T'_7] - \text{Prob}[T'_8]| = |\text{Prob}[\text{Coll}_7] - \text{Prob}[\text{Coll}_8]| \leq 2^{-s(\lambda)+1}$$

since this is the statistical distance between the two distributions.

We can now bound  $\text{Prob}[\text{Coll}_8]$ . Recall that in game  $G'_8$ , the challenge  $\langle u_1^*, u_2^*, v^* \rangle$  is a random triple over  $\langle g_1 \rangle \times \langle g_1 \rangle \times \langle g_1 \rangle$ , so we can turn the game  $G'_8$  into a collision finder for the hash function  $\mathcal{H}$ . Using this, we see

$$\text{Prob}[\text{Coll}_8] \leq \text{Adv}_{\text{UOWHF}}^{\text{Gen}}(\lambda).$$

**Closing argument.** From the above, we can see that  $\text{Prob}[T_7] = 1/2$ ; moreover, we proved  $\text{Prob}[\text{Coll}] \leq 2^{-s(\lambda)+1} + \text{Adv}_{\text{UOWHF}}^{\text{Gen}}(\lambda)$  through the sequence of games  $G'_6, G'_7, G'_8$ . Based on this, our sequence of games allows us to conclude

$$\left| \text{Prob}[T_0] - \frac{1}{2} \right| \leq \text{Adv}_{\text{DDH}}^{\text{Gen}}(\lambda) + \text{Adv}_{\text{UOWHF}}^{\text{Gen}}(\lambda) + (Q_1 + Q_2 + 3) \cdot 2^{-s(\lambda)+1}.$$

This is negligible in  $\lambda$  since  $Q_1, Q_2$  are polynomial bounded in  $\lambda$  and  $\text{Adv}_{\text{DDH}}^{\text{Gen}}(\lambda)$  and  $\text{Adv}_{\text{UOWHF}}^{\text{Gen}}(\lambda)$  are assumed to be negligible in  $\lambda$  (by Assumptions 1 and 2). This proves the following theorem.

**Theorem 11.3.1.** *The CS public-key cryptosystem satisfies IND-CCA2 security under Assumptions 1 and 2.*

## 12 Privacy Primitives

The cryptographic primitives we have discussed thus far are insufficient to solve problems concerning the privacy of a system's users. Indeed, privacy in the sense of anonymity is a different requirement than any previous issue we have dealt with. In light of this, we turn our discussion toward hiding the identity of participants in a system. In this section, we focus on two important cryptographic primitives involved in privacy applications: blind signatures and mix-servers.

### 12.1 Blind Signatures

A blind signature is a scheme that allows a signer to authenticate a document without having any information about the document itself. The two primary objectives of a blind signature are unforgeability and blindness, where blindness refers to the property that any information in a document is kept private from a signer. In this section, we introduce the Chaum blind signature, which was introduced by David Chaum in 1982.

The Chaum blind signature is based on the Full-Domain Hash RSA signature scheme. We saw in Section ?? that a message  $M$  is signed by  $H(M)^d = \sigma$  and a message-signature pair  $(M, \sigma)$  is verified by testing if  $\sigma^e \equiv H(M) \pmod{n}$ . We now adapt this for a blind signature.

**Definition 12.1.1.** Let  $e$  be a prime,  $M$  a string,  $n$  an RSA modulus, and  $H$  a hash function. A two-move **blind signature scheme** is a tuple of algorithms (Blind, Sign, Unblind, Verify) such that

- The blinding algorithm **Blind**: Given  $M$ , a random number  $r \stackrel{\text{r}}{\leftarrow} \mathbb{Z}_n$ , and the public key  $pk = (n, e)$ , output  $M' = r^e H(M) \bmod n$ .
- The signing algorithm **Sign**: Given a blinded message  $M'$  and the secret key  $sk = d$ , output a digital signature  $\sigma' = (M')^d \bmod n$ .
- The algorithm **Unblind**: Given a signature  $\sigma'$  of a blinded message  $M'$ , compute  $\sigma = \sigma' r^{-1} \bmod n$ .
- The verification algorithm **Verify**: For any  $(M, \sigma)$ , test if  $\sigma^e = (\sigma' r^{-1})^e = M' r^{-e} = H(M) \bmod n$ . If equality holds, return **True**=1; otherwise return **False**=0.

Figure ?? describes the general formulations of these algorithms for a user  $\mathcal{U}$  and a signer  $\mathcal{S}$ .

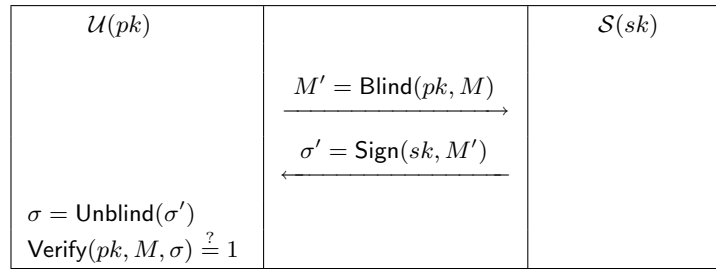


Figure 20: The Chaum blind signature generation.

**Definition 12.1.2.** The **blindness property** states that for any two messages  $M_0, M_1$  and all algorithms  $\mathcal{A}$

$$|\text{Prob}[\mathcal{A}(\text{Blind}(pk, M_0)) = 1] - \text{Prob}[\mathcal{A}(\text{Blind}(pk, M_1)) = 1]| \leq \varepsilon.$$

When  $\varepsilon$  is negligible, we say there is **statistical blindness**; when  $\varepsilon = 0$ , there is **perfect blindness**, i.e. the distributions of  $\text{Blind}(pk, M_0)$  and  $\text{Blind}(pk, M_1)$  are statistically or perfectly indistinguishable.

**Theorem 12.1.1.** *The Chaum blind signature satisfies statistical blindness.*

*Proof.* If there are two messages  $M_0$  and  $M_1$  such that  $r^e H(M_0) \equiv (r')^e H(M_1) \bmod n$  for  $r, r' \in \mathbb{Z}_n$ , then  $r \equiv r' (H(M_1)/H(M_0))^{1/e} \bmod n$ . This uses the assumption that  $H(M_i) \in \mathbb{Z}_n^*$  and the fact that  $a \equiv b \bmod n$  if  $a^e \equiv b^e \bmod n$ . The latter we get from  $\text{gcd}(n, e) = 1$ . ■

### E-Cash System Based on the Chaum Blind Signature

When buying something with paper money or checks, a merchant can hold the money and verify it is legitimate. As society has moved away from using tangible money, new methods have arisen to validate electronic transactions. Systems such as **e-cash** are now used to instigate a third party in a transaction to ensure both the buyer and merchant are honest and protected.

Consider the following three-party scenario: the bank, the bank user Alf, and the merchant Mark. Suppose Alf decides to buy something at Mark's shop. He has an account with the bank, so he goes and identifies himself as an authentic account holder. He hands the bank an envelope containing a serial number, which the bank signs without opening. The bank has a certain denomination designated as an **e-coin**, say \$10. They then take \$10 from Alf's account and add an e-coin to the

pool- a common collection area where Alf's e-coin is indistinguishable from any other. Meanwhile, Alf walks down the street to Mark's shop, picks out what he wants and hands Mark the signed envelope. In order to complete the transaction, Mark goes to the bank to verify that the serial number and signature are good. The bank takes an e-coin from the pool and gives \$10 to Mark.

First note that Alf is free to make as many copies of the envelope's signature as he wishes. The bank however, only accepts a given signature once, rendering any duplicates useless. Second, the value of an e-coin is pre-determined. In order to purchase something, Alf may need the bank to sign multiple envelopes in order to have adequate e-coins in the pool. Here the pool ensures that the bank cannot determine who or how much Alf paid.

We can represent this in a more formal manner. Suppose the bank publishes  $e, n$ , and  $H$  to all merchants. Let  $\mathcal{U}$  be a user,  $\mathcal{M}$  a merchant, and  $\mathcal{B}$  the bank. To withdraw an e-coin from an account,

1.  $\mathcal{U}$  identifies himself to the bank.
2.  $\mathcal{U}$  selects a random number `rnd`.
3.  $\mathcal{U}$  selects  $r \xleftarrow{\mathcal{R}} \mathbb{Z}_n$ .
4.  $\mathcal{U}$  calculates  $y = r^e H(M || \text{rnd}) \bmod n$  and sends  $y$  to  $\mathcal{B}$ .
5.  $\mathcal{B}$  moves the value of an e-coin from  $\mathcal{U}$ 's account and puts an e-coin in the pool.
6.  $\mathcal{B}$  replies to  $\mathcal{U}$  with  $\sigma = y^{1/e} \bmod n$ .
7.  $\mathcal{U}$  computes  $\text{coin} = r^{-1} \sigma \bmod n$  and outputs  $\langle \text{rnd}, \text{coin} \rangle$  as the e-coin.

In Step 4,  $r$  is used for blinding. By dividing by  $r$  in Step 7, the blinding is removed.

To make a payment,  $\mathcal{U}$  gives  $\langle \text{rnd}, \text{coin} \rangle$  to  $\mathcal{M}$ . Before providing the purchased services,  $\mathcal{M}$  checks that  $(\text{coin})^e \equiv H(M || \text{rnd}) \bmod n$ . If it is valid,  $\mathcal{M}$  submits  $\langle \text{rnd}, \text{coin} \rangle$  to the bank through an authenticated channel.

When the bank receives  $\langle \text{rnd}, \text{coin} \rangle$ , it checks that the pair is good, then looks to see if it already exists in the deposited e-coin database. If it has not been used, the bank moves the value of the e-coin from the pool to  $\mathcal{M}$ 's account.

In this setting, it is impossible to link a payment to the corresponding withdrawal.

### E-Voting Scheme Based on the Chaum Blind Signature

We can similarly construct an electronic voting (*e-voting*) scheme based on the Chaum blind signature. The three parties involved are the administrator  $A$ , the voters  $V_i$ , and the counter  $C$ .

The administrator  $A$  checks that  $V_i$  has the right to vote. Then, using some blinding factor,  $V_i$  blinds his ballot  $v_i$  into  $v'_i$  and asks  $A$  to produce the signature  $\sigma'_i$  for  $v'_i$ . To cast his ballot,  $V_i$  retrieves the unblinded signature  $\sigma_i$  of  $v_i$  from  $\sigma'_i$ . He verifies that  $(v_i, \sigma_i)$  is a valid ballot-signature pair using the administrator's verification key. If `Verify` returns `True`,  $V_i$  sends  $(v_i, \sigma_i)$  to the counter  $C$  through an anonymous communication channel.

The counter  $C$  uses the administrator's verification key to check the ballot-signature pair  $(v_i, \sigma_i)$  and then adds it to a list. After all voters vote,  $C$  counts the votes, publishes the list, and announces the results.

This scheme prevents the administrator from seeing who voted for whom. It does not however, disguise this information from the counter. To solve this privacy issue, we introduce a mix-server.

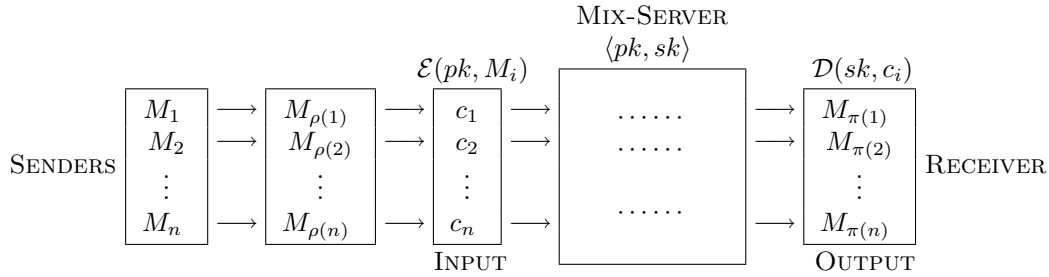


Figure 21: A mix-server with random permutations  $\rho, \pi$  on  $\{1, \dots, n\}$ .

## 12.2 Mix-Servers

A *mix-server* or mixer is a network that shuffles a group of messages and passes them to the receiver in a permuted order. The primary purpose for such a mechanism is to provide “sender privacy”; that is, it ensures that the entity receiving the mix-server’s output cannot discern who transmitted a message. In some sense, the mix-server separates a set of senders from a single receiver and attempts to conceal the sender-message relationships.

The messages themselves may reveal some information about the input vector or permutation. If each message is authenticated, the task of the mix-server becomes Sisyphean. More generally, if the receiver can obtain information about the input vector, he may be able to discern the sender-message relationship. For instance, suppose all messages contain the answer to a petition: one sender answers **Yes** and all others answer **No**. No matter how the mixer permutes the messages, the receiver can successfully identify the sender he suspects to have answered **Yes**.

Given that the receiver (the adversary in this setting) gets each message, the mix-server’s security goal does not include data privacy. We want a public key mechanism to eliminate private channels between the mix-server and the senders. To achieve this, we use the ElGamal encryption scheme:

$$\begin{aligned}
 \mathcal{G}(1^\lambda) : \quad & \langle pk, sk \rangle \leftarrow \mathcal{G}(1^\lambda) \\
 & x \xleftarrow{\mathcal{R}} \mathbb{Z}_m, h = g^x \bmod p \\
 & pk = \langle \langle p, m, g \rangle, h \rangle \\
 & sk = x \\
 \\
 \mathcal{E}(pk, M) : \quad & M \in \langle g \rangle \\
 & r \xleftarrow{\mathcal{R}} \mathbb{Z}_m \\
 & \text{compute } G = g^r \bmod p, H = h^r M \bmod p \\
 & \text{output } \langle G, H \rangle \\
 \\
 \mathcal{D}(sk, g, H) : \quad & \text{compute } M = H/G^x \bmod p \\
 & \text{output } M
 \end{aligned}$$

Figure ?? illustrates how a mix-server interacts with the senders and receiver. An adversary can view the original messages  $\{M_i\}$ , the “input” wires  $\{c_i\}$ , and the “output” wires  $\{M_{\pi(i)}\}$ . His goal is to find the relationship between the input and output wires, i.e. the permutation  $\rho$  that assigns messages to users.

In addition to constructing a public-key scheme, we have several objectives in creating an effective mix-server.

1. We want to relax our trust assumptions on the mixer for situations when the mixer is adversarial. Specifically, we must ensure that a mixer outputs authentic messages, not those of its



own creation.

2. The receiver should be unable to make correlations between messages and senders.
3. We want to prohibit a mix-server and receiver from forming a coalition; otherwise all privacy is lost. One method is to use multiple mix-servers: privacy is more convincing when there is evidence of at least one non-adversarial mixer.
4. And finally, we want the ability to scale a receiver to a mix-server, creating a second mix-server. Repeatedly doing so composes a sequence of servers and improves privacy.

### Zero-Knowledge Proof of Correct Decryption

Suppose the receiver asks a mixer to prove it properly decrypted the ciphertext  $c = \langle G, H \rangle$ . The mixer could publish  $M$  and demonstrate it knows the secret key  $x$  such that  $G^x = H/M$ . This amounts to providing a proof of knowledge for the discrete logarithm of  $\log_G(H/M)$ , which is an unconvincing argument. Indeed, a malicious mixer could fabricate a message by choosing a random  $x'$  and setting  $M = G^{x'}/H$ . A stronger argument requires that the mixer prove the equality of two logarithms:  $\log_G(H/M) \stackrel{?}{=} \log_g h$ . The corresponding zero-knowledge proof is given in Figure ??.

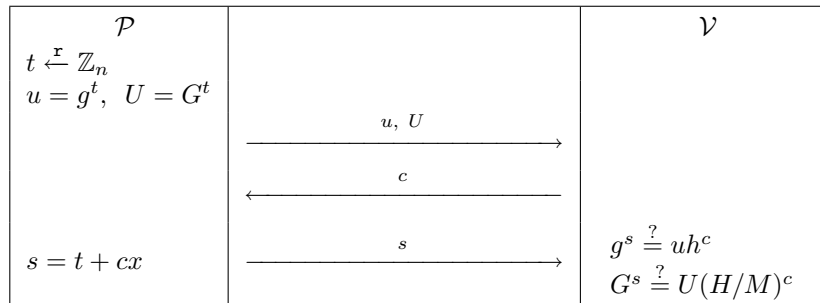


Figure 22: A zero-knowledge proof that verifies if  $x = \log_g h = \log_G H/M$ .

Once we verify that the mix-server correctly decrypted a ciphertext, we want to prove that each output message  $M'_i$  is the correct decryption of one of the inputs; namely that there is a permutation  $\pi$  such that for each ciphertext  $c_i = \langle G_i, H_i \rangle$  with  $\mathcal{D}(sk, c_i) = M'_i$ , there is some  $j$  for which  $M'_i = M_{\pi(j)}$ . Define the language

$$\mathbf{L} = \left\{ \langle c_1, \dots, c_n, M'_1, \dots, M'_n \rangle : \text{there is } (\pi, sk) \text{ such that } \mathcal{D}(sk, c_i) = M'_{\pi(i)} \text{ for all } 1 \leq i \leq n \right\}.$$

$\mathbf{L}$  is in  $NP$ . Given  $\pi$  and  $sk$ , one can verify if a string is valid in polynomial-time. Our membership problem can therefore be reduced to a Hamiltonian cycle. While this works, the resulting graph can be quite large. A more effective method is to first define the language

$$\mathbf{L}' = \left\{ \langle c_1, \dots, c_n, M'_1, \dots, M'_n \rangle : \text{there is an } sk \text{ such that for all } i \text{ there is a } j \text{ with } \mathcal{D}(sk, c_i) = M'_j \right\}$$

This language can also be written as

$$\mathbf{L}' = \bigwedge_{i=1}^n \left( \bigvee_{j=1}^n \mathcal{D}(sk, c_i) = m_j \right).$$

Note that  $\mathbf{L} \subsetneq \mathbf{L}'$  since  $\mathbf{L}'$  does not require that all original messages appear, i.e., in  $\mathbf{L}'$ , a mixer could set  $\mathcal{D}(sk, c_i) = M'_j$  for all  $i$  and a fixed  $j$ .  $\mathbf{L}$  mandates that all decrypted messages be distinct,

and therefore all plaintexts must be distinct (no ciphertext can be decrypted in two ways). Although the two languages are different, we can use the disjunction of two zero-knowledge proofs on  $\mathbf{L}'$  to verify a solution.

We require that all plaintexts be distinct. Depending on the system, this may not naturally occur. We can enforce distinctness however, by inserting randomness into each input. For example, consider the modified encryption  $\mathcal{E}(pk, M) = (g^r, h^r(M\|s))$  for a random string  $s$ . The receiver now must check that all opened plaintexts have different  $s$ -components and accept only if this is the case. Assuming  $s$  is sufficiently long (at least 128 bits), a collision happens with probability  $2^{-64}$  by the birthday paradox. It follows that a collision is likely to reveal a misbehaved mixer.

### Serial Composition of $n$ Mixers

For added security, we can cascade a set of  $n$  mixers to encrypt messages layer by layer. This introduces the problem of encrypting ciphertext, which is inefficient in the public-key setting because it causes expansion in the message size. Here we present an efficient solution to this dilemma.

Suppose in a set of  $n$  mixers, Mixer  $i$  has the public key  $pk_i = \langle \langle p, m, g \rangle, h_i \rangle$  and secret key  $sk_i = x_i$ . A message  $M$  is encrypted as  $\langle g^r, (h_1 h_2 \cdots h_n)^r (M\|s) \rangle = \langle G, H \rangle$ . If Mixer 1 decrypts the ciphertext  $\langle G, H \rangle$  as  $\langle G, H/G^{x_1} \rangle$ , the output is a valid compound ciphertext for mixers 2, 3,  $\dots$ ,  $n$ . The problem is that we fixed the first part of the ciphertext, potentially making the input-output relationship traceable: the mixer is not really permuting. We have to allow mixers to re-encrypt each message. The decryption of Mixer  $i$  is then,

$$\mathcal{D}_i(x_i, G, H) = \left\langle g^{r'} G, \left( \frac{H}{G^{x_i}} \prod_{j=i+1}^n h_j^{r'} \right) \right\rangle.$$

To prove the mix-server is now faithfully following the protocol, we consider the case of 2 mixers. Suppose the first mixer decrypts  $\langle G, H \rangle = \langle g^r, (h_1 h_2)^r M \rangle$  and outputs  $\langle G', H' \rangle = \langle G g^{r'}, h_2^{r'} H / G^{x_1} \rangle$ . If the mixer can simultaneously prove three logs,  $g^{x_1} = h_1$ ,  $g^{r'} = G'/G$ , and  $h_2^{r'} / G^{x_1} = H'/H$ , it successfully proves it is faithful.

## 13 Distributing Trust

### 13.1 Secret sharing

In a *secret sharing scheme*, a set of players possess pieces of information that combine to form a common “secret”. Unless the number of involved players exceeds a certain threshold, no one can reconstruct or obtain *any* information about the secret.

Consider a simple example. Suppose a dealer  $D$  wants to distribute a secret  $S \in \mathbb{Z}_q$  to a set of players  $P_i$  for  $i = 1, \dots, n$ . The dealer randomly selects  $n - 1$  numbers  $s_1, \dots, s_{n-1} \stackrel{\times}{\leftarrow} \mathbb{Z}_q$  and sets the  $n$ th as  $s_n = S - s_1 - \dots - s_{n-1} \bmod q$ .  $D$  then distributes  $s_i$  to  $P_i$  as his share of the secret.

In the special case of  $n = 2$  players,  $s_1 + s_2 = S$ , where  $s_1 \stackrel{\times}{\leftarrow} \mathbb{Z}_q$  and  $s_2 = S - s_1 \bmod q$ . The probability distributions of  $s_1, s_2$  are the same: both  $s_1$  and  $s_2$  are randomly selected from  $\mathbb{Z}_q$  so neither  $P_1$  nor  $P_2$  can recover  $S$  without the help of the other player. In a similar fashion, one can show that any  $n - 1$  players cannot recover a secret shared by  $n$  players.

### 13.2 Shamir’s Secret Sharing Scheme

Define  $p(X) = a_0 + a_1 X + \dots + a_{t-1} X^{t-1}$  for  $a_i \in \mathbb{Z}_q$ . If we know  $t$  points  $(z_0, y_0), (z_1, y_1), \dots, (z_{t-1}, y_{t-1})$ , where  $y_i = p(z_i)$ , we can build the system

$$\begin{bmatrix} 1 & z_0 & \cdots & z_0^{t-1} \\ 1 & z_1 & \cdots & z_1^{t-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & z_{t-1} & \cdots & z_{t-1}^{t-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{t-1} \end{bmatrix} \equiv \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{t-1} \end{bmatrix} \pmod{q}. \quad (6)$$

Denote (??) by  $ZA \equiv Y \pmod{q}$ . If the determinant of  $Z$  is nonzero, we can solve  $A = Z^{-1}Y$  and find the coefficients of  $p(X)$  using Lagrange interpolation. This method forms the basis for **Shamir's secret sharing scheme**:

Suppose there are  $n$  shareholders and a secret threshold of  $t$ , i.e. fewer than  $t$  shareholders cannot recover any of the secret. The dealer  $D$  defines  $p(X)$  such that the constant term  $a_0 = p(0)$  is the secret and all other coefficients are randomly selected:  $a_1, \dots, a_{t-1} \xleftarrow{r} \mathbb{Z}_q$ .  $D$  distributes  $s_i = p(i)$  to each shareholder  $P_i$  for  $i = 1, \dots, n$ .

When  $t$  players meet, they can solve a system similar to (??). Denote this subset of players as  $\{P'_j\}$  for  $j = 1, \dots, t$ , where each of the  $t$  contributing players has the corresponding share  $s'_j$ . Using Lagrange interpolation, they can obtain  $a_0 = \lambda_1 s'_1 + \dots + \lambda_t s'_t$ , where each  $\lambda_j$  is a publicly constructible Lagrange coefficient. Note that with this method,  $t - 1$  players cannot compute any information about  $a_0$ , but  $t$  players can retrieve the entire secret.

### 13.3 Distributing Decryption Capabilities

We now return to the ElGamal encryption scheme with public key  $pk = \langle \langle p, m, g \rangle, h \rangle$  and secret key  $sk = x$  such that  $h = g^x$ . Using a secret sharing scheme, we can disperse the secret key among  $n$  players such that any  $t$  (or more) can together obtain  $x$  and decrypt the ElGamal ciphertext. The secret  $x$  however, can only be used once: after its reconstruction it is available to all players. We now introduce **threshold decryption**, which enables  $x$  to be reused.

Let  $\langle G, H \rangle = \langle g^r, h^r M \rangle$  be an ElGamal ciphertext with  $h = g^x$  and  $x = p(0)$ , where  $p(X)$  is defined as in Section ???. Distribute  $s_i = p(i)$  to the  $i$ th shareholder  $P_i$  for  $i = 1, \dots, n$ .

When a group of  $t$  players  $P'_j$ ,  $j = 1, \dots, t$ , decide to decrypt  $\langle G, H \rangle$ , each  $P'_j$  publishes  $G_j = G^{s'_j}$ .

$$\begin{array}{c|c|c|c} P'_1 & P'_2 & \cdots & P'_t \\ s'_1 & s'_2 & \cdots & s'_t \\ \hline G_1 & G_2 & \cdots & G_t \end{array}$$

Using the Lagrange coefficients, the  $t$  players compute

$$\begin{aligned} G_1^{\lambda_1} G_2^{\lambda_2} \cdots G_t^{\lambda_t} &= G^{\lambda_1 s'_1} \cdots G^{\lambda_t s'_t} \\ &= G^{p(0)} \\ &= G^x \\ &= g^{rx} \\ &= h^r \end{aligned}$$

This implies that the plaintext  $M$  can be obtained by

$$M = H / G_1^{\lambda_1} G_2^{\lambda_2} \cdots G_t^{\lambda_t}.$$

### Application to E-Voting

**Definition 13.3.1.** Given two groups  $(X, +)$  and  $(Y, \cdot)$ <sup>8</sup>, a *group homomorphism* is a function  $\varphi: X \rightarrow Y$  such that for all  $\alpha, \beta \in X$ ,

$$\varphi(\alpha + \beta) = \varphi(\alpha) \cdot \varphi(\beta).$$

This implies  $\varphi$  must also preserve identities and inverses.

Define  $(X, +)$  to be the plaintext group and  $(Y, \cdot)$  to be the ciphertext group where  $\cdot$  is some group operation on ciphertexts. Given  $C_1, C_2 \in Y$  such that  $C_1 = \mathcal{E}(pk, M_1)$  and  $C_2 = \mathcal{E}(pk, M_2)$  for  $M_1, M_2 \in X$ , we call  $\mathcal{E}$  a *homomorphic encryption function* if

$$\mathcal{E}(pk, M_1 + M_2) \approx C_1 \cdot C_2^9.$$

As an example, consider a variation of ElGamal encryption. Let  $\mathbb{Z}_m$  be the plaintext group (under addition modulo  $m$ ) and let  $\langle \mathbb{Z}_p^*, \mathbb{Z}_p^* \rangle$  be the ciphertext group for a prime  $p$ . Take any ciphertext  $\langle G, H \rangle = \langle g^r, h^{r+M} \rangle$  with  $M \in \mathbb{Z}_m$ . To decrypt  $\langle G, H \rangle$ , compute  $H/G^x = h^M$  and search through all possible choices of  $\{h^{M_1}, h^{M_2}, \dots, h^{M_m}\}$ . This encryption function satisfies the desired homomorphic properties, although it is only efficient for small  $m$ .

Using the homomorphic properties of ElGamal, we can create a threshold decryption scheme for an e-voting system with  $n$  voters. Suppose the  $i$ th voter provides identification and submits  $\langle G(i), H(i) \rangle = \langle g^{r_i}, h^{r_i+M_i} \rangle$ , where  $M_i \in \{0, 1\}$  (No=0, Yes=1). Let  $A = \sum r_i$  and  $B = \sum M_i$  for  $i = 1, \dots, n$ . Observe that

$$\left\langle \prod_{i=1}^n G(i), \prod_{i=1}^n H(i) \right\rangle = \langle g^A, h^A h^B \rangle$$

is a ciphertext that encrypts the number of voters who answered **Yes**:  $B = \sum_{i=1}^n M_i$ .

The question is if  $\langle G(i), H(i) \rangle$  is a valid ciphertext for  $M_i \in \{0, 1\}$ . Note that if  $M_i = 0$ , then  $\log_g h = \log_{G(j)} H(j)$  and if  $M_i = 1$ , then  $\log_g h = \log_{G(j)} H(j)/h$ . Since  $g, h, G(i), H(i)$  are public information,  $V_i$  can prove one of these discrete logarithms in the disjunction of two zero-knowledge proofs.

We now turn our attention to (publicly) verifiable secret sharing and dealer-less secret sharing, in which special versions of secret sharing are applicable.

## 13.4 Publicly Verifiable Secret Sharing

In a threshold decryption scheme, problems can occur when one of the  $t$  contributing players incorrectly publishes his part. The  $t - 1$  honest players are blocked from the secret while the malicious player is able to reconstruct it. To prevent this, we instigate a third party or judge to which each shareholder must prove the validity of their share. We first present a solution that weakens the security of the scheme.

Suppose the dealer publishes

$$\langle g^{a_0}, g^{a_1}, \dots, g^{a_{t-1}} \rangle = \langle V_0, V_1, \dots, V_{t-1} \rangle.$$

If  $s_i$  corresponds to the  $i$ th shareholder, notice

$$\begin{aligned} V_0 V_1^i \cdots V_{t-1}^{i^{t-1}} &= g^{a_0} g^{a_1 i} \cdots g^{a_{t-1} i^{t-1}} \\ &= g^{p(i)} \\ &= g^{s_i}, \end{aligned}$$

<sup>8</sup>For clarity, we write  $X$  using additive notation and  $Y$  using multiplicative notation, even though the group operations on  $X$  and  $Y$  may very well be different than the usual addition and multiplication operations.

<sup>9</sup>The symbol  $\approx$  represents an identical distribution

so the  $i$ th shareholder can present the publicly verifiable value  $g^{s_i}$  to a judge. This solves our dilemma, but weakens the security because an adversary can similarly construct a value and present it to a judge.

A better solution uses zero-knowledge proofs. Note that  $G, g, g^{s_i}$  are public and the  $i$ th player claims  $G_i = G^{s_i}$ . One can verify this by showing  $\log_G G_i = \log_g g^{s_i}$ . To prove he is honest, the  $i$ th player then provides a zero-knowledge proof for the discrete logarithms, as discussed in previous sections.

### 13.5 Distributing the Dealer

We have a major issue in the previous schemes in that the dealer knows the secret, in addition to each player's share. We want to avoid any secret constructions and all-knowing parties at the beginning of the process. To accomplish this, we distribute the dealer's responsibilities among the shareholders.

Let  $P_i$  select  $S_i$  as his secret.  $P_i$  uses his private polynomial  $p_i(X) = a_{i,0} + a_{i,1}X + \dots + a_{i,t-1}X^{t-1}$  to compute  $S_{i,j} = p_i(j)$ . He gives  $S_{i,j}$  to  $P_j$  as the  $j$ th share of  $S_i$ .  $P_i$  then publishes  $V_{i,0}, V_{i,1}, \dots, V_{i,t-1}$ .

The  $j$ th shareholder now collects  $S_{1,j}, S_{2,j}, \dots, S_{n,j}$  from the other players and computes his share as  $s_j = \sum_{i=1}^n S_{i,j}$ . The shared secret is  $\sum_{i=1}^n p_i(0)$  and the verification values are  $V_k = \prod_{i=1}^n V_{i,k}$  for  $k = 0, \dots, t-1$ . Note that no non-sharing entity knows the secret, but the secret can still be recovered by any attempt of  $t$  users.

## 14 Broadcast Encryption

In a system with users in  $[N] = \{1, \dots, N\}$ , a **broadcast encryption scheme** enables a party called the Center to send a message to a select subgroup of  $[N]$ , excluding whomever he chooses. A common example of broadcast encryption is visible in cable companies. They distribute various cable packages through a common network such that subscribers can only obtain the package they paid for.

Let  $R \subseteq [N]$  be the subset of users excluded from a transmission. We define the **enabled set**  $E = [N] \setminus R$  to be the intended recipients of a ciphertext. In this section, we assume all encryption schemes are symmetric:  $\mathcal{E}(k, M)$  and  $\mathcal{D}(k, c)$ .

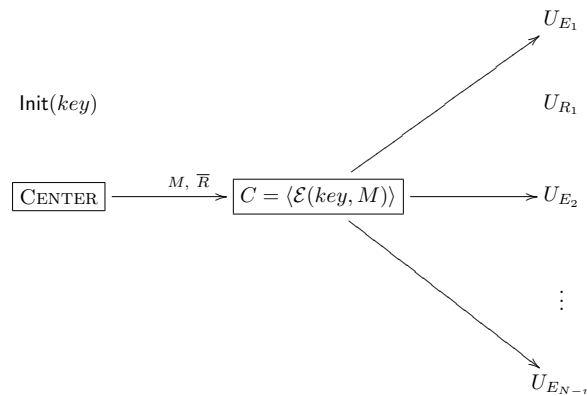


Figure 23: A broadcast encryption scheme where users  $U_{E_i}$  can decrypt  $M$ , but  $U_{R_j}$  cannot, with  $E_i \in E$  and  $R_i \in R$ .  $\bar{R}$  denotes the description of  $R$ .

After the network initializes the system with  $\text{Init}(\text{key})$ , the Center sends the ciphertext  $C$  to all enabled users  $U_{E_i}$  with  $E_i \in E$  in such a way that no  $U_{R_i}$  can decrypt  $M$  with  $R_i \in R$ .

The trivial solution is to give every user a distinct key  $k_i$ . If  $|R| = r$ , the Center encrypts  $M$  with each of the  $N - r$  appropriate keys. The ciphertext  $C = \langle \mathcal{E}(k_{E_1}, M), \dots, \mathcal{E}(k_{E_{N-r}}, M) \rangle$  then has length  $N - r$ . To decrypt the message, each enabled user tries their key on every entry in the ciphertext. While this works, it requires  $C$  to be very long. One of our primary goals is to minimize the length of the ciphertext.

Consider an alternate solution. Instead of giving a key to each user, give a key to every subset of  $[N]$ . This is called the **poset**  $P$  of  $[N]$ <sup>10</sup>. To send a message, the Center now only needs the key corresponding to the set  $E$ .

**Example.** Take  $N = 3$  and  $R = \{2\}$  as seen in Figure ?? . Each node corresponds to a key, so User 1 has keys for  $\{1\}$ ,  $\{1, 2\}$ ,  $\{1, 3\}$ ,  $\{1, 2, 3\}$  and User 3 has keys for  $\{3\}$ ,  $\{1, 3\}$ ,  $\{2, 3\}$ ,  $\{1, 2, 3\}$ . Note that  $\{1, 3\}$  is the only set whose key is available to Users 1 and 3, but not User 2. This is also the set  $E$ . In Figure ??, nodes containing  $R$  have been left open to indicate that the Center cannot use the corresponding keys. While there are four solid nodes, only the node for  $E = \{1, 3\}$  is needed.

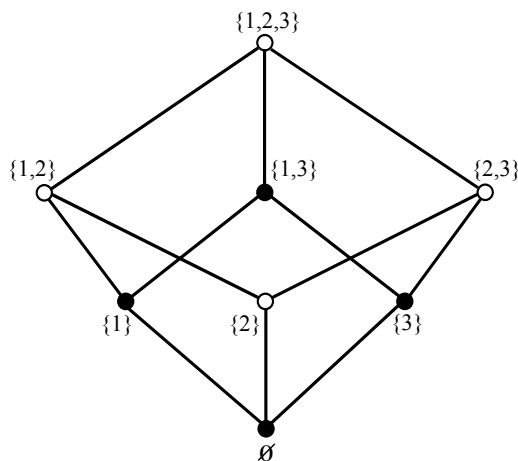


Figure 24: All possible subsets of  $[N]$  for  $N = 3$  and  $R = \{2\}$ .

Using the poset enables the Center to send a ciphertext with a minimal number of keys. The largest possible ciphertext has length  $\lambda + r \log r$  where  $\lambda = |\mathcal{E}(\text{key}, M)|$ . This solution is therefore most helpful when  $r$  is small. Recall that  $|P| = 2^N$ , so each user needs  $2^{N-1}$  keys.

The method of using posets is inefficient in that each user must store the maximum number of keys. Our goal is now to find a medium between minimizing the number of keys required by the Center to encrypt a message and minimizing the number of keys required by each user to decrypt a message.

## 14.1 Complete Binary Trees

One effective way to minimize both the ciphertext size and user memory space is to use complete binary trees.

**Definition 14.1.1.** A **complete binary tree** (CBT) is a tree diagram in which every level is full and all levels have the same depth<sup>11</sup>. Any CBT has  $2^t$  leaves for some  $t \in \mathbb{Z}^+$ .

<sup>10</sup>The term poset comes from “partially ordered set”

<sup>11</sup>The **depth** of a tree is the number of steps it takes to move from a leaf to the root. Visually, this is the number of segments in the path from leaf to root.

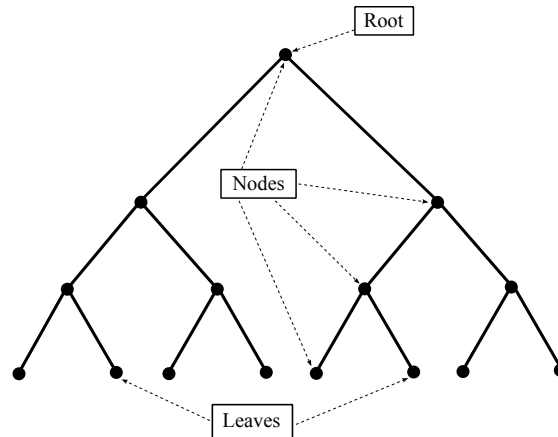


Figure 25: A complete binary tree with  $N = 8$  leaves and depth 3.

**Lemma 14.1.1.** *In a complete binary tree with  $N$  leaves, there are  $2N - 1$  nodes.*

*Proof.* We prove this by induction on  $N$ . When  $N = 1$ , it holds trivially that there is  $2 - 1 = 1$  node. Assume if  $N = 2^t$ , there are  $2(2^t) - 1 = 2^{t+1} - 1$  nodes.

Now take any CBT with  $N = 2^{t+1}$  leaves. By removing the root, we divide the tree into two symmetric complete binary subtrees, each with  $2^t$  leaves. By our inductive assumption, each subtree has  $2^{t+1} - 1$  nodes. The entire tree then has

$$2(2^{t+1} - 1) + 1 = 2(2^{t+1}) - 1 = 2N - 1$$

nodes. ■

Figure ?? illustrates how  $N = 8$  users can be represented in a CBT, where each leaf denotes one user. Each node then corresponds to a key assigned to the subset of users hanging below that node.

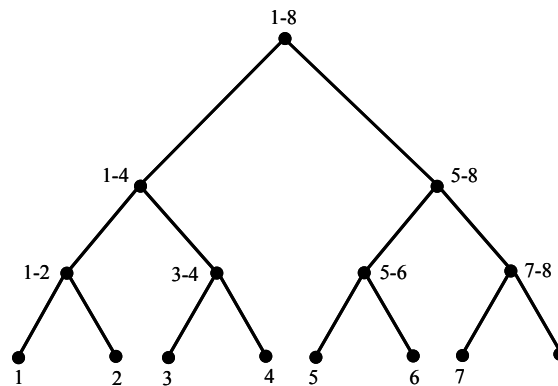


Figure 26: A complete binary tree representing  $N = 8$  users in a broadcast encryption scheme.

It is clear that any CBT has depth  $\log N$  (base 2), so each user needs  $1 + \log N$  keys. To communicate with all  $N$  users, the Center only needs the root's key to encrypt his message.

In an ideal situation, the members of  $R$  form a complete subtree, so  $r = 2^x$ . When excluding  $R$  from a transmission, the center refrains from using the root of  $R$ 's subtree, and any ancestral node the subtree hangs from. Each user then needs a maximum of  $\log N - \log r = \log(N/r)$  keys. This is illustrated in Figure ??, where open nodes denote excluded keys.

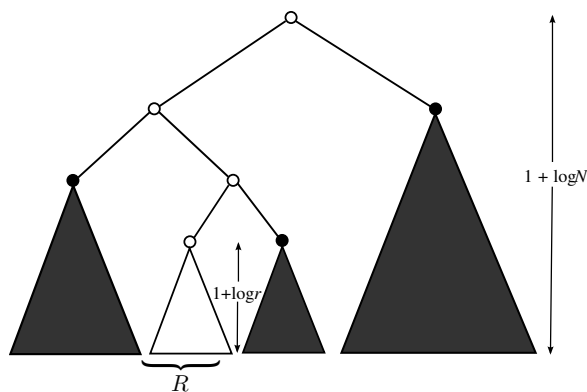


Figure 27: Excluding  $R$  when  $R$  forms a single complete subtree.

As  $R$  grows, or disperses into different subtrees, users may need up to  $r \log(N/r)$  keys. This implies that the ciphertext will also have length between  $r \log N$  and  $r \log(N/r)$ . This is a superior method for broadcast encryption.

## 15 Elliptic Curve Cryptography

Thus far, we have focused only on cryptosystems built over multiplicative groups. In this section, we discuss new systems and revisit old systems built over elliptic curves, which are additive Abelian groups.

In recent years, there has been a huge movement toward understanding elliptic curves for cryptographic purposes. There are two readily apparent benefits for using curves over modular groups. The first is their cost efficiency. A 200-bit elliptic curve provides the same security as a 1,000-bit modular group. The second reason is that there are no known ways to generalize the attacks against the discrete logarithm problem on a modular group to an attack on an elliptic curve; although this may be due to our present lack of understanding of elliptic curves.

### 15.1 Elliptic Curves

While it requires very advanced mathematics to formally define an elliptic curve, they are easily dealt with abstractly, based on their additive structure and pairing properties. Perhaps the simplest definition for an *elliptic curve* is the set of solutions to a generalized cubic equation.

We are primarily interested in curves over a finite field  $F$ . When  $F$  is not of characteristic<sup>12</sup> 2 or 3, we can write any elliptic curve as a plane algebraic curve defined by<sup>13</sup>

$$Y^2 = X^3 + aX + b, \quad (7)$$

where  $a, b \in F$ .

As an additive group, any two points on a curve add to produce a third point on the curve. This form of addition is geometrically interpreted differently from the previous notion of vector addition. On an elliptic curve, adding points  $P_1$  and  $P_2$  is viewed as projecting a line between  $P_1$  and  $P_2$  to form the point of intersection with the curve. The point of intersection is the resulting point  $P_3$ .

<sup>12</sup>The *characteristic* of a ring is the smallest positive integer  $n$  such that  $n \cdot 1 = 0$ . If no such  $n$  exists, the ring is said to have characteristic zero. Fields always have prime characteristic.

<sup>13</sup>Equation (??) is a special form of the Weierstrass equation for curves not over fields of characteristic 2 or 3.



Using this idea, the additive inverse of a point  $P = (x, y)$  is  $-P = (x, -y)$ . The identity 0 is called the **point at infinity**, and therefore sometimes also denoted as  $\infty$ . For any curve  $E$ , the property must hold that  $P_1 + P_2 + P_3 = 0$  for any points  $P_1, P_2, P_3 \in E$ .

We have three possibilities when adding  $P_1 = (x_1, y_1)$  and  $P_2 = (x_2, y_2)$ :

$$P_1 + P_2 = \begin{cases} 0 & x_1 = x_2 \text{ or } y_1 = -y_2 \\ P_2 = P_2 + P_1 & P_1 = 0 \\ P_3 = (x_3, y_3) & \text{otherwise.} \end{cases}$$

Formally,  $P_3$  is calculated as

$$x_3 = m^2 - x_1 - x_2 \quad \text{and} \quad y_3 = y_1 + m(x_3 - x_1),$$

where the slope  $m$  between  $P_1$  and  $P_2$  is

$$m = \begin{cases} (y_2 - y_1)/(x_2 - x_1) & P_1 \neq P_2 \\ (3x_1^2 + a)/(2y_1) & P_1 = P_2. \end{cases}$$

When  $P_1 = P_2$ , the line connecting the “two” points is the line tangent to  $P_1$ .

## 15.2 Bilinear Maps

One key concept in elliptic curve cryptography is that of a pairing. A **pairing** is a function that uses two points on a curve as inputs, and outputs an element of some multiplicative Abelian group. Two very useful pairings are the Weil and Tate pairing functions, which we will not discuss in detail [?].

One of the major pairing-based constructions is that of the bilinear map.

**Definition 15.2.1.** Let  $\mathbb{G}_1$  and  $\mathbb{G}_2$  be two groups of prime order  $q$ <sup>14</sup>. An admissible **bilinear map**

$$e: \mathbb{G}_1 \times \mathbb{G}_1 \longrightarrow \mathbb{G}_2$$

satisfies three properties:

- **Bilinearity:** For all  $P, Q \in \mathbb{G}_1$  and all  $a, b \in \mathbb{Z}_q^*$ , it holds that  $e(aP, bQ) = e(P, Q)^{ab}$ .
- **Non-Degeneracy:**  $e$  does not map all pairs in  $\mathbb{G}_1 \times \mathbb{G}_1$  to the identity in  $\mathbb{G}_2$ . That is, for all nonzero  $P \in \mathbb{G}_1$ ,  $e(P, P) \neq 1$ . This implies that if  $P$  is a generator of  $\mathbb{G}_1$ ,  $e(P, P)$  generates  $\mathbb{G}_2$ :  $\langle e(P, P) \rangle = \mathbb{G}_2$ .
- **Computability:** There is an effective algorithm to compute  $e$ .

The existence of such an admissible bilinear map is proven by the Weil and Tate pairings [?]. In the following examples, we take  $\mathbb{G}_1$  to be an additive group of points on an elliptic curve, and  $\mathbb{G}_2$  to be a finite field.

Using the pairing function, we can prove several complexity implications [?].

**Theorem 15.2.1.** *The discrete logarithm problem in  $\mathbb{G}_1$  is no harder than the discrete logarithm problem in  $\mathbb{G}_2$ .*

*Proof.* Take  $P, Q \in \mathbb{G}_1$ . The discrete logarithm problem in  $\mathbb{G}_1$  amounts to finding an  $a \in \mathbb{Z}_q$  such that  $Q = aP$ . Let  $\alpha = e(P, P)$  and  $\beta = e(P, Q)$ . By the bilinearity of  $e$ ,  $\beta = e(P, P)^a = \alpha^a$ . By non-degeneracy,  $\alpha$  and  $\beta$  both have prime order  $q$ , so this establishes a reduction of the discrete logarithm problem into  $\mathbb{G}_2$ . ■

<sup>14</sup>For distinction, we write  $\mathbb{G}_1$  using additive notation and  $\mathbb{G}_2$  using multiplicative notation. In general, the group operations of  $\mathbb{G}_1$  and  $\mathbb{G}_2$  may be different from the usual addition and multiplication.

**Theorem 15.2.2.** *The decisional Diffie-Hellman problem is easy in  $\mathbb{G}_1$ .*

*Proof.* To solve DDH, one must distinguish between the distributions of  $\langle P, aP, bP, cP \rangle$  and  $\langle P, aP, bP, abP \rangle$  where  $a, b, c \stackrel{r}{\leftarrow} \mathbb{Z}_q^*$  and  $P$  is a random point in  $\mathbb{G}_1$ . We can build a distinguisher that breaks the DDH assumption as follows:

1. Compute  $g_1 = e(aP, bP)$  and  $g_2 = e(P, cP)$ .
2. If  $g_1 = g_2$ , then the tuple is of the form  $\langle P, aP, bP, abP \rangle$ .

To see this, notice

$$g_1 = e(aP, bP) = e(P, P)^{ab} = e(P, abP).$$

So when  $c = ab \pmod q$ ,  $g_1 = g_2$ . The distinguisher can therefore tell the tuples apart.  $\blacksquare$

### 15.3 Bilinear Diffie-Hellman Assumption

While DDH is not hard in  $\mathbb{G}_1$ , the computational Diffie-Hellman problem can still be hard; that is, given a random  $\langle P, aP, bP \rangle$ , it is believed to be hard to find  $abP \in \mathbb{G}_1$ . A helpful variant of CDH is the bilinear Diffie-Hellman problem.

**Definition 15.3.1.** The goal of the *bilinear Diffie-Hellman problem* (BDH) in  $\langle \mathbb{G}_1, \mathbb{G}_2, e \rangle$  is to compute  $e(P, P)^{abc}$  when given  $\langle P, aP, bP, cP \rangle$  for some  $a, b, c \in \mathbb{Z}_q^*$ . An adversary  $\mathcal{A}$  has advantage  $\varepsilon$  in solving BDH if

$$\text{Adv}^{\mathcal{A}} = \text{Prob}[\mathcal{A}(P, aP, bP, cP) = e(P, P)^{abc}] \geq \varepsilon.$$

The hardness of BDH depends on an appropriate choice of security parameters in  $\mathbb{G}_2$  [?].

**Definition 15.3.2.** A *BDH parameter generator* is a randomized, polynomial-time computable algorithm  $\mathcal{G}$  that on input  $\lambda$ , outputs a prime  $q$ , the description of two groups  $\mathbb{G}_1, \mathbb{G}_2$  of order  $q$ , and the description of an admissible bilinear map  $e: \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ . We denote the output of  $\mathcal{G}$  on the security parameter  $\lambda$  as  $\mathcal{G}(1^\lambda) = \langle q, \mathbb{G}_1, \mathbb{G}_2, e \rangle$ .

**Definition 15.3.3.** A BDH parameter generator  $\mathcal{G}$  is said to satisfy the *bilinear Diffie-Hellman assumption* provided

$$\text{Adv}_{\mathcal{G}}^{\mathcal{A}}(\lambda) = \text{Prob} \left[ \mathcal{A}(q, \mathbb{G}_1, \mathbb{G}_2, e, P, aP, bP, cP) = e(P, P)^{abc} : \begin{array}{l} \langle q, \mathbb{G}_1, \mathbb{G}_2, e \rangle \leftarrow \mathcal{G}(1^\lambda), \\ P \stackrel{r}{\leftarrow} \mathbb{G}_1, a, b, c \stackrel{r}{\leftarrow} \mathbb{Z}_q^* \end{array} \right]$$

is negligible for all PPT algorithms  $\mathcal{A}$ . Informally, the BDH assumption considers it hard to compute  $e(P, P)^{abc}$  when given  $\langle P, aP, bP, cP \rangle$ .

### 15.4 One-Round, 3-Part Key Agreement Scheme

The most immediate way to use elliptic curves is in the Diffie-Hellman key exchange protocol and ElGamal. As we just showed, the security of DDH fails under the pairing function, so we must modify our protocols under the BDH assumption. One interesting example of a key agreement that benefits from this modification is the one-round, 3-party key agreement scheme introduced by Joux in 2000. This was the first scheme enabling a third honest party to contribute to the exchange and legally obtain a key. It was also the first scheme based on Diffie-Hellman that required only one round of exchanged data.

There are three parties,  $A, B, C$  with secrets  $a, b, c$  respectively. Assume there is an admissible bilinear map  $e$  between the groups  $\mathbb{G}_1 \times \mathbb{G}_1$  and  $\mathbb{G}_2$  available to all parties and  $P$  is a known generator of  $\mathbb{G}_1$ . In the Joux scheme,

1.  $A$  sends  $aP$  to  $B, C$ .

2.  $B$  sends  $bP$  to  $A, C$ .
3.  $C$  sends  $cP$  to  $A, B$ .
  - Steps 1-3 are performed in one round of parallel message exchanges.
4.  $A$  computes  $e(bP, cP)^a = e(P, P)^{abc}$ .
5.  $B$  computes  $e(aP, cP)^b = e(P, P)^{abc}$ .
6.  $C$  computes  $e(aP, bP)^c = e(P, P)^{abc}$ .
  - Steps 4-6 are performed simultaneously.

Using this protocol, all parties obtain the common key  $K = e(P, P)^{abc}$ , which is secure assuming  $BDH$  is hard

## 15.5 Identity-Based Encryption

Identity-based encryption (IBE) was first imagined by Shamir in 1984. His original motivation was to develop a public-key encryption scheme that enabled users to send messages encrypted with publicly known strings. For example, Alice can send an encrypted email to Bob, using his email address as the public key. To decrypt his message, Bob needs to obtain his private decryption key from an independent third party, called the Private Key Generator (PKG). One perk is that Alice can send her message before Bob obtains his private key. She can also encrypt her email using both Bob's email address and a time frame; Bob then cannot open the email until the specified time.

In 2001, Boneh and Franklin proposed the first practical IBE scheme using the Weil pairing  $\hat{e}$  on an elliptic curve [?]. In the concrete model,  $\mathbb{G}_1$  is a subgroup of additive points of an elliptic curve over a finite field,  $E/\mathbb{F}_p$ , and  $\mathbb{G}_2$  is a subgroup of the multiplicative group of a finite field  $\mathbb{F}_{p^2}$ . The security of the IBE system is proven to be IND-ID-CCA under the BDH assumption; we define IND-ID-CCA security later in Section ??.

### The BasicPub IBE Scheme

Boneh and Franklin's full IBE scheme is a variant of ElGamal. To prove IND-ID-CCA security, Boneh and Franklin reduced their scheme to several simpler encryption schemes first and proved the security of those models. The simplest of the reduced encryption schemes is BasicPub, which is defined as follows.

**Key-generation.** For input  $1^\lambda$ , run  $\mathcal{G}$  to obtain the system parameters  $\langle q, \mathbb{G}_1, \mathbb{G}_2, \hat{e} \rangle$  where  $q$  is a  $\lambda$ -bit prime,  $\mathbb{G}_1$  and  $\mathbb{G}_2$  are groups of order  $q$ , and  $\hat{e}$  is an admissible bilinear map  $\hat{e}: \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ . Choose a random generator  $P$  of  $\mathbb{G}_1$ . Set  $s \xleftarrow{\mathcal{R}} \mathbb{Z}_q^*$  as the system-wide secret key and  $P_{pub} = sP$  as the system-wide public key. Choose  $Q_{id} \xleftarrow{\mathcal{R}} \mathbb{G}_1^*$ . Let  $H_2$  be a cryptographic hash function  $H_2: \mathbb{G}_2 \rightarrow \{0, 1\}^n$  for some  $n$ . The public key is  $\langle q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, n, P, P_{pub}, Q_{id}, H_2 \rangle$ . The private key is  $d_{id} = sQ_{id} \in \mathbb{G}_1^*$ .

**Encryption.** Given a message  $M \in \{0, 1\}^n$ , choose  $r \xleftarrow{\mathcal{R}} \mathbb{Z}_q^*$  and return  $C = \langle rP, M \oplus H_2(g^r) \rangle$ , where  $g = \hat{e}(Q_{id}, P_{pub}) \in \mathbb{G}_2^*$ .

**Decryption.** Given a ciphertext  $C = \langle U, V \rangle$ , use  $d_{id}$  to compute  $V \oplus H_2(\hat{e}(d_{id}, U)) = M$ .

Suppose we encrypt a message  $M$  with public key  $id$  to obtain  $\langle rP, M \oplus H_2(g^r) \rangle$  for some  $r \in \mathbb{Z}_q^*$ . Call this ciphertext  $C = \langle U, V \rangle$ . The decryption algorithm works as follows,

$$\begin{aligned}
V \oplus H_2(\widehat{e}(d_{id}, U)) &= V \oplus H_2(\widehat{e}(sQ_{id}, rP)) \\
&= V \oplus H_2(\widehat{e}(Q_{id}, P)^{rs}) \\
&= V \oplus H_2(\widehat{e}(Q_{id}, sP)^r) \\
&= V \oplus H_2(g^r) \\
&= M \oplus H_2(g^r) \oplus H_2(g^r) \\
&= M
\end{aligned}$$

BasicPub is not a true IBE scheme since each identity is an arbitrary random string. Here an adversary only attacks one user, he can never obtain the secret key for other users. This however lays the groundwork for the next scheme, BasicIdent, which is IBE.

Before we discuss BasicIdent, let us examine the security of BasicPub. While Boneh and Franklin proved a much stronger result, here we show that BasicPub is IND-CPA secure under the bilinear decisional Diffie-Hellman assumption (BDDH).

**Definition 15.5.1.** For any  $\langle q, \mathbb{G}_1, \mathbb{G}_2, e \rangle$  produced from  $\mathcal{G}$ , the  **bilinear decisional Diffie-Hellman assumption**  claims that an adversary cannot distinguish between tuples of the form  $\langle P, aP, bP, cP, e(P, P)^{abc} \rangle$  and  $\langle P, aP, bP, cP, e(P, P)^d \rangle$  with more than a negligible probability when  $a, b, c, d \stackrel{\mathcal{R}}{\leftarrow} \mathbb{Z}_q^*$ .

**Proof of Security.** As in Section ??, we can model the security as a sequence of games.

#### The IND-CPA Game.

Game $G_0$ on Input $1^\lambda$	Random Variables
1. $\langle q, \mathbb{G}_1, \mathbb{G}_2, \widehat{e} \rangle \leftarrow \mathcal{G}(1^\lambda; \rho)$	$\rho \leftarrow \text{Coins}$
2. $P_{pub} \leftarrow sP$	$s \stackrel{\mathcal{R}}{\leftarrow} \mathbb{Z}_q^*$
3. $Q_{id} \stackrel{\mathcal{R}}{\leftarrow} \mathbb{G}_1^*$	
4. $\langle \text{aux}, M_0, M_1 \rangle \leftarrow \mathcal{A}^{H_2(\cdot)[q_{H_2}]}(\text{stage}_1, q, \mathbb{G}_1, \mathbb{G}_2, \widehat{e}, n, P, P_{pub}, Q_{id}, H_2; \rho_1)$	$\rho_1 \stackrel{\mathcal{R}}{\leftarrow} \text{Coins}_1$
5. $U^* \leftarrow rP$	$r \stackrel{\mathcal{R}}{\leftarrow} \mathbb{Z}_q^*$
6. $V^* \leftarrow M_b \oplus H_2(\widehat{e}(Q_{id}, P_{pub})^r)$	$b \stackrel{\mathcal{R}}{\leftarrow} \{0, 1\}$
7. $b^* \leftarrow \mathcal{A}(\text{stage}_2, \text{aux}, U^*, V^*; \rho_2)$	$\rho_2 \stackrel{\mathcal{R}}{\leftarrow} \text{Coins}_2$
8. if $b = b^*$ return 1 else return 0	

The random oracle  $H_2(\cdot)$  is queried  $q_{H_2}$  times during Step 3. When given a query  $X_i$ , it answers with  $H_2(X_i) = H_i$ .

**Modification 1.** We first modify Step 3 in  $G_0$  to obtain  $G_1$ .

3. $Q_{id} \leftarrow tP$	$t \stackrel{\mathcal{R}}{\leftarrow} \mathbb{Z}_q^*$
---------------------------	---

Since  $Q_{id}$  is random in  $G_0$  and  $t$  is random in  $G_1$  (and therefore  $tP$  is also random) the probability distributions of  $[Q_{id}]_0$  and  $[Q_{id}]_1$  are uniform, proving  $\text{Prob}[T_0] = \text{Prob}[T_1]$ .

**Modification [2].** Here we modify  $H_2$  as a random oracle in Step 4 to obtain  $G_2$ .  $H_2$  now maintains a list of all queries and answers,  $List_{H_2} = \{\langle X_i, H_i \rangle\}$  for  $i = 1, \dots, q_{H_2}$ , and operates as

1. Given  $X_i$  such that  $\langle X_i, H_i \rangle \in List_{H_2}$  for some  $H_i \in \{0, 1\}^n$ , return  $H_2(X_i) = H_i$ .
2. Given  $X_i$  such that  $\langle X_i, H \rangle \notin List_{H_2}$  for some  $H \in \{0, 1\}^n$ , choose  $H_i \stackrel{\mathcal{R}}{\leftarrow} \{0, 1\}^n$ , enter  $\langle X_i, H_i \rangle \in List_{H_2}$ , and return  $H_2(X_i) = H_i$ .

Again, this is a trivial adjustment so  $\text{Prob}[T_1] = \text{Prob}[T_2]$ .

**Modification 3.** We modify Step 6 now to obtain  $G_3$ .

6. $V^* \leftarrow M_b \oplus H_2(\widehat{e}(P, P)^d)$	$b \xleftarrow{r} \{0, 1\}, d \xleftarrow{r} \mathbb{Z}_q^*$
---	--

Note that the variables  $r, s, t, d$  are not explicitly used anywhere in  $G_2$  or  $G_3$ . Moreover, for any parameters  $\langle q, \mathbb{G}_1, \mathbb{G}_2, \widehat{e} \rangle$  produced by  $\mathcal{G}$ , the triple  $\langle P, rP, sP, tP, e(P, P)^{rst} \rangle$  is distributed as a BDDH tuple in game  $G_2$  and as a random tuple in game  $G_3$ . From the third game-playing lemma then, we have that  $|\text{Prob}[T_2] - \text{Prob}[T_3]| \leq \text{Adv}_{\text{BDDH}}^A(\lambda)$ .

**Modification 4.** For our last modification, we alter Step 6 of  $G_3$  to obtain  $G_4$ .

6. $V^* \xleftarrow{r} \mathbb{G}_2^*$	
--	--

Define  $F$  to be the event that  $\widehat{e}(Q_{id}, P_{pub})^r \in \text{List}_{H_2}$ . Clearly, if  $\mathcal{A}$  does not make a query about  $\widehat{e}(Q_{id}, P_{pub})^r$ ,  $\text{Prob}[T_3 \cap \neg F] = \text{Prob}[T_4 \cap \neg F]$ . Assuming  $\#\mathbb{G}_1 \geq 2^\lambda$ , the first game-playing lemma yields

$$|\text{Prob}[T_3] - \text{Prob}[T_4]| \leq \frac{qH_2}{2^\lambda}.$$

**Closing Argument.** Through the previous series of modifications, it is now clear that  $\text{Prob}[T_5] = 1/2$  ( $b$  is never used before Step 8). To conclude the proof, note that the sequence of games reveals

$$\left| \text{Prob}[T_0] - \frac{1}{2} \right| \leq \text{Adv}_{\text{BDDH}}^A(\lambda) + \frac{qH_2}{2^\lambda}.$$

Since both  $\text{Adv}_{\text{BDDH}}^A(\lambda)$  and  $\frac{qH_2}{2^\lambda}$  are assumed to be negligible in  $\lambda$ , this is also negligible. This proves the following theorem.

**Theorem 15.5.1.** *The BasicPub encryption scheme satisfies IND-CPA security under the bilinear decisional Diffie-Hellman assumption in the random oracle model.*

### The BasicIdent IBE Scheme

Proving IND-CPA for BasicPub is a comparatively weak result for an IBE system. The adversary is not allowed to become a member of the system and himself have an identity. We now expand to IND-ID-CPA security, where an adversary may make private key extraction queries to the PKG, call it  $K$ . Suppose  $K$  has the system-wide secret key  $sk_A$ , then we can model IND-ID-CPA security as the following game.

#### The IND-ID-CPA Game.

Game $G_{\text{IND-ID-CPA}}$ on Input $1^\lambda$	Random Variables
1. $\langle \text{param}, sk_A \rangle \leftarrow \mathcal{G}(1^\lambda; \rho)$	$\rho \xleftarrow{r} \{0, 1\}^\lambda$
2. $\langle \text{aux}, id, M_0, M_1 \rangle \leftarrow \mathcal{A}^{K(sk_A, \cdot)}(\text{stage}_1, \text{aux}, \text{param}; \rho_1)$	$\rho_1 \xleftarrow{r} \{0, 1\}^\lambda$
3. if $id \in \text{Computed}$ or $M_0 = M_1$ then stop	
4. $C \leftarrow \mathcal{E}(\text{param}, id, M_b)$	$b \xleftarrow{r} \{0, 1\}$
5. $b^* \leftarrow \mathcal{A}^{K^{-id}(sk_A, \cdot)}(\text{stage}_2, \text{aux}; \rho_2)$	$\rho_2 \xleftarrow{r} \{0, 1\}^\lambda$
11. if $b = b^*$ return 1 else return 0	

We now shift our attention to a stronger IBE system, BasicIdent:

**Setup.** For input  $1^\lambda$ , run  $\mathcal{G}$  to obtain the system parameters  $\langle q, \mathbb{G}_1, \mathbb{G}_2, \widehat{e} \rangle$  where  $q$  is a  $\lambda$ -bit prime,  $\mathbb{G}_1$  and  $\mathbb{G}_2$  are groups of order  $q$ , and  $\widehat{e}$  is an admissible bilinear map  $\widehat{e}: \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ . Choose a

random generator  $P$  of  $\mathbb{G}_1$ . Set  $s \xleftarrow{\mathcal{R}} \mathbb{Z}_q^*$  as the system-wide secret key and  $P_{pub} = sP$  as the system-wide public key. Let  $H_1$  and  $H_2$  be a cryptographic hash functions such that  $H_1: \{0, 1\}^* \rightarrow \mathbb{G}_1^*$  and  $H_2: \mathbb{G}_2 \rightarrow \{0, 1\}^n$  for some  $n$ .

**Extract.** For any string  $id \in \{0, 1\}^*$ , compute  $Q_{id} = H_1(id) \in \mathbb{G}_1^*$  and set  $d_{id} = sQ_{id}$  as the private key.

**Encryption.** Given a message  $M \in \{0, 1\}^n$ , compute  $Q_{id}$ , choose  $r \xleftarrow{\mathcal{R}} \mathbb{Z}_q^*$ , and return  $C = \langle rP, M \oplus H_2(g_{id}^r) \rangle$ , where  $g_{id} = \widehat{e}(Q_{id}, P_{pub}) \in \mathbb{G}_2^*$ .

The primary difference between BasicIdent and BasicPub is that BasicIdent uses a well-defined hash function to determine  $Q_{id}$  based on the user's identity; whereas BasicPub selects  $Q_{id}$  at random. Because of this, an IND-ID-CPA attack on BasicIdent can be reduced to an IND-CPA attack on BasicPub.

**Theorem 15.5.2.** *BasicIdent is IND-ID-CPA secure under the BDH assumption in the random oracle model.*

One way to prove security is to assume  $H_1$  behaves as a random oracle. Bohen and Franklin do this in the Random Oracle Model, using the IND-CPA security of BasicPub and the random oracle behavior of  $H_1$ . It is also possible to rely on the weaker BDH assumption with the random oracle.

### The IND-ID-CCA Security Model

The standard security model for an IBE system is IND-ID-CCA, where the adversarial goal is the chosen ciphertext attack. Under such a model, we assume the adversary can obtain the private key  $d_{id_j}$  for any identity  $id_j$  not under attack. The adversary can also make decryption queries  $\langle id_j, C_j \rangle$  on ciphertexts  $C_j$  of his choosing. Both the private key extraction queries and decryption queries can be made adaptively, depending on previous results. In general, IND-ID-CCA can be modeled as the following two-phase game:

#### The IND-ID-CCA Game.

Game $G_{\text{IND-ID-CCA}}$ on Input $1^\lambda$	Random Variables
1. $\langle \text{param}, sk_A \rangle \leftarrow \mathcal{G}(1^\lambda; \rho)$	$\rho \xleftarrow{\mathcal{R}} \{0, 1\}^\lambda$
2. $\langle \text{aux}, id, M_0, M_1 \rangle \leftarrow \mathcal{A}^{K(sk_A, \cdot), \text{Dec}(K(sk_A, \cdot), \cdot)}(\text{stage}_1, \text{aux}, \text{param}; \rho_1)$	$\rho_1 \xleftarrow{\mathcal{R}} \{0, 1\}^\lambda$
3. if $id \in \text{Computed}$ or $M_0 = M_1$ then stop	
4. $C \leftarrow \mathcal{E}(\text{param}, id, M_b)$	$b \xleftarrow{\mathcal{R}} \{0, 1\}$
5. $b^* \leftarrow \mathcal{A}^{K^{-id}(sk_A, \cdot), \text{Dec}^{-id, C}(K(sk_A, \cdot), \cdot)}(\text{stage}_2, \text{aux}; \rho_2)$	$\rho_2 \xleftarrow{\mathcal{R}} \{0, 1\}^\lambda$
11. if $b = b^*$ return 1 else return 0	

This is the ideal model for security because it allows the adversary to make two separate rounds of queries; one round before submitting the challenge values and one round after receiving the challenge ciphertext  $C$ . After  $\mathcal{A}$  receives  $C$ , he may alter  $C$  to make  $C'$ . Since  $C' \neq C$ , he may then make decryption queries in Step 5. If the scheme is IND-ID-CCA secure, the adversary cannot obtain any additional information about  $b$  from decrypting the modified ciphertext.

**Definition 15.5.2.** A public key identity based encryption scheme is IND-ID-CCA secure if for all PPT adversaries  $\mathcal{A}$ ,

$$\text{Adv}_{\text{IND-ID-CCA}}^{\mathcal{A}}(\lambda) = \left| \text{Prob}[b = b^*] - \frac{1}{2} \right|$$

is negligible.

## 16 Simulation Based Security

In previous sections, we defined security by playing a sequence of games with an adversary. One problem with this method is our lack of ability to evaluate a game's sufficiency. We have no guarantee all important cases are covered; we may not yet know what cases are important.

An alternate way to evaluate security is through a simulation-based method. The goal is now to define the ideal a given protocol is intended to approximate and find a way to analyze if the protocol realizes that ideal. This model, in its fullest generality, was developed by Canetti in 2001 [?].

If a protocol realizes its ideal functionality, the agreement between the protocol and its ideal is maintained through a variety of settings. To capture these diverse scenarios, we introduce the notion of an *environment* that describes a particular setting involving an application of the protocol. While simulation-based security is a very broad area of study, this section we only address it in the narrow context of key exchange protocols.

In general, an environment  $Z$  creates parties  $P_1, P_2, \dots$  and enables each to perform a protocol.  $Z$  also creates an adversary  $\mathcal{A}$  and allows him to do bad things, including corrupting parties and using their capabilities. For example, in Figure ??  $\mathcal{A}$  corrupts  $P_2$ , so now  $P_2$  shares all of his information with  $\mathcal{A}$ .

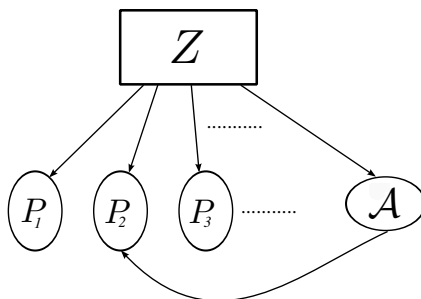


Figure 28: An environment  $Z$  with users  $P_i$  and adversary  $\mathcal{A}$ .

Any key exchange protocol is defined by the description of two types of parties: initiators  $I$  and responders  $R$ . The environment dictates what and when  $I$  and  $R$  do anything. The session ID  $sid$  contain the names of both  $I$  and  $R$ , and any auxiliary information about their communications (e.g. a session number, a timestamp, etc.). The template for the operations of these programs is as follows.

- Whenever program  $I$  is given input (`InitiatorStart, sid`),  $I$  initiates a key exchange protocol with a party  $R$ , whose name is determined by  $sid$ .
- When the responding party  $R$  is contacted by the initiating party, it produces the output (`ResponderAck, sid`) in acknowledgment.
- When the responder  $R$  receives input (`ResponderStart, sid`), it follows the protocol and responds to the initiator.
- After a number of rounds of communication, the initiator and responder terminate and return the output (`Key, sid, k`). (Depending on the particular protocol instantiation, either party may terminate first.)

In an execution of the protocol, the environment  $Z$  can create multiple parties of this description and witness the course of execution. The environment also creates an adversary  $\mathcal{A}$  that intercepts all communication between  $I$  and  $R$  and, if allowed by  $Z$ , corrupts and takes full control of any of these parties. The environment and adversary communicate in an arbitrary manner.

All of the above describes what happens in the “real world” environment. We assume in the real world that all communication between parties takes place through the adversary  $\mathcal{A}$ . We further assume  $\mathcal{A}$  cannot modify messages from an honest party. This amounts to using an authenticated communication link between the parties. Under a suitable assumption, a digital signature can be used to create such links, but this is not the focus of this section.

Next we define the “ideal world” environment. In an ideal world, there is an ideal functionality  $\mathcal{F}_{KE}$  which operates as follows.

- Upon receiving an input (`InitiatorStart`,  $sid$ ) from party  $I$ , verify  $sid = (I, R, sid')$  for some identity  $R$ , record  $I$  as active, record  $R$  as the responder, and send a public delayed output<sup>15</sup> (`ResponderAck`,  $sid$ ) to  $R$ .
- Upon receiving (`ResponderStart`,  $sid$ ) from  $R$ , verify  $R$  is recorded as the responder, record  $R$  as active, and then notify the adversary.
- After receiving a message (`Key`,  $sid, P, k^*$ ) from the adversary, for  $P \in \{I, R\}$  do:
  - If  $P$  is active and neither  $I$  nor  $R$  is corrupt: if there is no recorded (`Key`,  $sid, k$ ), randomly choose  $k$  and record (`Key`,  $sid, k$ ). Output (`Key`,  $sid, k$ ) to  $P$ .
  - Else, if  $P$  is active and either  $I$  or  $R$  is corrupt, output (`Key`,  $sid, k^*$ ) to  $P$ .
  - Else,  $P$  is not active: do nothing.

## 16.1 The 2DH Key Exchange Protocol

To exhibit the differences between the real and ideal worlds, recall the Diffie-Hellman key exchange protocol from Section 5.1. Here we refer to this as the 2DH key exchange protocol. In the real world,  $Z$  first creates an adversary  $\mathcal{A}$  and instructs him to pass all information through  $Z$ . In this case, the adversary is less trivial and performs actions beyond being merely “passed through”. Referencing Figure ??, the protocol now operates as follows.

1.  $Z$  sends (`InitiatorStart`,  $(I, R)$ ) to  $I$  to initiate a key exchange.
2.  $I$  randomly generates  $x$  and sends  $a = g^x$  to  $R$ .
3.  $\mathcal{A}$  may block the message, ending the protocol; otherwise  $a$  passes to  $R$ .
4.  $R$  sends (`ResponderAck`,  $(I, R)$ ) to  $Z$  to acknowledge he was contacted by  $I$ .
5.  $Z$  sends (`ResponderStart`,  $(I, R)$ ) to  $R$ .
6.  $R$  randomly generates  $y$  and sends  $k = a^y$  to  $Z$ .
7.  $R$  sends  $b = g^y$  back to  $I$ .
8.  $\mathcal{A}$  may block the message, ending the protocol; otherwise  $b$  passes to  $I$ .
9.  $I$  sends  $k = b^x$  back to  $Z$ .

Now consider the operation of the environment in an ideal world. Here the initiator and responder become transparent entities and their operations are substituted by the ideal functionality  $\mathcal{F}_{KE}$ . To distinguish between  $\mathcal{F}_{KE}$ ’s functionality as  $I$  and  $R$ , we write  $\mathcal{F}_{KE}(I)$  and  $\mathcal{F}_{KE}(R)$  respectively. Note however, that there is really only one function. In the real world simulation, the environment is in full communication with the adversary. To convince  $Z$  of the existence of an adversary in

<sup>15</sup>A *public delayed output* is a mechanism to return an output to parties after “asking” an adversary for permission. This gives the adversary the opportunity in the ideal world to block the output, should it wish to do so.



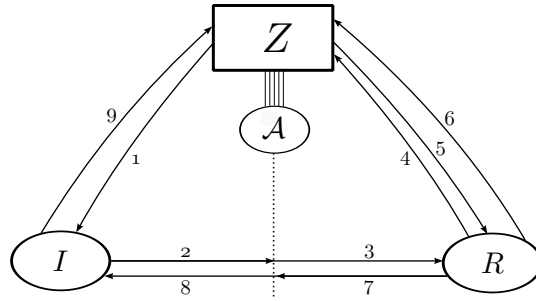


Figure 29: The 2DH key exchange protocol in the real world simulation.

the ideal world, we create an adversary simulator  $S$  that runs the adversary  $\mathcal{A}$  internally. The ideal functionality still communicates with the adversary, but this is not the same adversary as it is restricted from corrupting other parties.

Using Figure ??, the 2DH key-exchange protocol in the ideal world is as follows.

1.  $Z$  sends (InitiatorStart,  $(I, R)$ ) to  $\mathcal{F}_{KE}(I)$  to initiate a key exchange.
2.  $S$  is activated through the public delayed output. It may perform actions to present a picture similar to the real world to the environment. For example,  $S$  may simulate  $I$  to randomly generate  $x$  and send  $a = g^x$  to  $R$ .
3.  $\mathcal{A}$  may block the message, ending the protocol; otherwise  $a$  passes to  $R$ . If  $a$  passes to  $R$ , then  $S$  decides that the public delayed output should go through and returns control to the functionality  $\mathcal{F}_{KE}$ .
4.  $\mathcal{F}_{KE}(R)$  sends (ResponderAck,  $(I, R)$ ) to  $Z$  to acknowledge he was contacted by  $\mathcal{F}_{KE}(I)$
5.  $Z$  sends (ResponderStart,  $(I, R)$ ) to  $\mathcal{F}_{KE}(R)$ .
6.  $\mathcal{F}_{KE}(R)$  randomly generates  $k^*$  and sends  $k^*$  to  $Z$ .
7.  $S$  is again activated through the public delayed output. It may perform actions to present a picture similar to the real world to the environment. For example,  $S$  may simulate  $R$  and generate  $y$ , compute  $b = a^y$  and send  $b$  to  $I$ .
8.  $\mathcal{A}$  may block the message, ending the protocol; otherwise  $b$  passes to  $I$ . If  $b$  passes to  $I$ , then  $S$  decides that the public delayed output should go through and returns control to the functionality  $\mathcal{F}_{KE}$ .
9.  $\mathcal{F}_{KE}(I)$  sends  $k^*$  back to  $Z$ .

**Definition 16.1.1.** We say an ideal key exchange is secure if for all real world adversaries  $\mathcal{A}$ , there exists an ideal world adversary  $S$  such that for all environments  $Z$ ,  $Z$  cannot tell the difference between the real world execution with  $\mathcal{A}$  and the ideal world execution with  $S$ .

The entire appeal of the above security definition is that any real world adversary is transformed into an ideal world adversary without the environment knowing the difference. This is meaningful as a security definition because real world adversaries can corrupt parties and do bad things without restriction; whereas ideal world adversaries may also do bad things, but only in prescribed ways defined in the ideal functionality. In the real world, the adversary is stronger than the parties. The reverse is true in the ideal world.

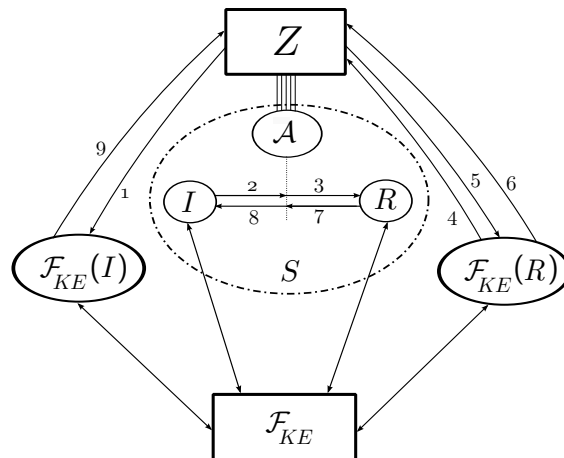


Figure 30: The 2DH key-exchange protocol in the ideal world simulation.

In 2DH,  $Z$  receives  $k = g^{xy}$  as the key in the real world, and a random  $k^*$  in the ideal world. Assuming no parties are corrupted, distinguishing between the two worlds amounts to solving DDH. While we showed in Section ?? that the Diffie-Hellman key exchange protocol was secure against passive adversaries, it does not remain so under simulation-based security. This does not necessarily mean the proof in Section ?? is incorrect, only that it solely considered eavesdropping adversaries.

To illustrate how 2DH fails under simulation-based security, consider what happens when  $Z$  allows the adversary to corrupt a party. The environment that attacks 2DH operates as follows.

1. After receiving the key from  $R$ , block the message between Steps 7 and 8. Tell  $\mathcal{A}$  to corrupt  $I$  and return the contents of  $I$  (namely  $x$ ).
2. If  $a = g^x$  and  $k = b^x$ , output 1 else 0.

In the real world,  $Z$  always outputs 1. In the ideal world,  $S$  has to prove it committed itself to some  $x'$  such that  $a = g^{x'}$  and  $k^* = b^{x'}$ . Since  $k^*$  is randomly generated, this is equivalent to breaking the discrete logarithm problem.

## 16.2 The 2mDH Key Exchange Protocol

We can modify 2DH so that it satisfies simulation-based security. The primary problem in the previous case is that  $x$  was stored in the initiator's memory. Now consider the 2mDH key exchange protocol in Figure ??.

Now should an environment corrupt the initiator (Alice) after receiving the key from the responder (Bob), no information can be obtained about  $x$ . This provides the indistinguishability necessary between the real and ideal worlds.

## References

- [1] V. Shoup, *A Computational Introduction to Number Theory and Algebra*, Cambridge University Press, Cambridge, UK, 2005. <http://www.shoup.net/ntb/>.
- [2] V. Shoup, *Sequences of Games: A Tool for Taming Complexity in Security Proofs*, 2006. <http://eprint.iacr.org/2004/332>.

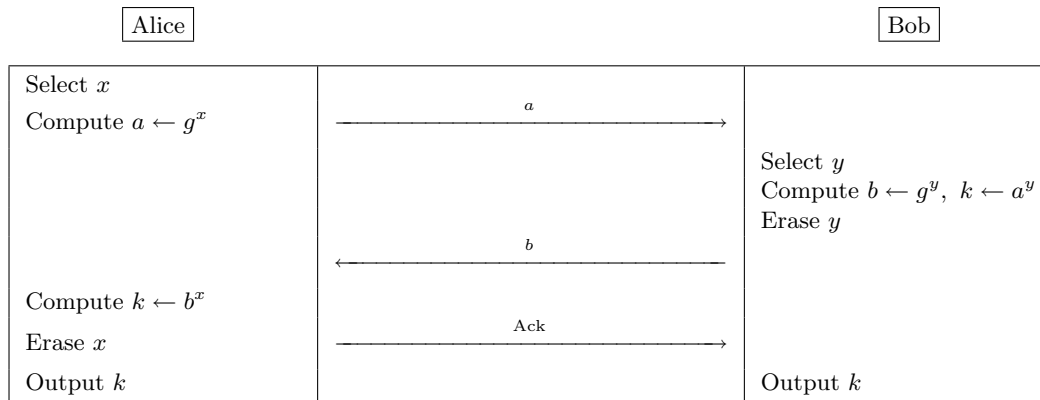


Figure 31: The 2mDH key exchange protocol.

- [3] M. Bellare and P. Rogaway, *Code-Based Game-Playing Proofs and the Security of Triple Encryption*, 2008. <http://eprint.iacr.org/2004/331>
- [4] D. Bohen, M. Franklin, *Identity-Based Encryption from the Weil Pairing*, SIAM J. of Computing, Vol. 32, No. 3, pp. 586-615, 2003.
- [5] R. Canetti, *Universally Composable Security: A New Paradigm for Cryptographic Protocols*, 2001. <http://eccc.hpi-web.de/pub/eccc/reports/2001/TR01-016/index.html>.
- [6] R. Canetti, R. Rivest, *Lecture 25: Pairing-Based Cryptography* 6.897 Special Topics in Cryptography, 2004.