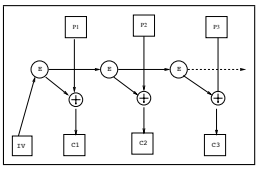


Aggelos Kiayias

Cryptography

Primitives and Protocols

Based on notes by G. Panagiotakos, S. Pehlivanoglu, O.S. Thyfronitis Litos, J. Todd, K. Samari, T. Zacharias, H.S. Zhou and P. Chaidos

Contents

1 Introduction

To begin discussing the basic properties of cryptography and illustrate the current state of the discipline we will consider a basic problem of trust related to coin tossing.

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.

To understand the solution, it is helpful to think conceptually how the problem can be solved using a box. Alice tosses her coin and places it in the box. This forces Alice to be consistent and prevents her from changing the result. Here the box constitutes a ***commitment*** mechanism. Although Bob still needs to open the box and check the outcome, by employing the box, both parties no longer need to be physically present simultaneously to toss a coin.

What can be the digital equivalent of a box? Let us consider the following. Suppose there is a pre-agreed upon mapping f that sends each of 0 and 1 to a set of objects at random. The mapping f will play the role of the box. 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 committing 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 α, Bob selects $b = 0$ with probability β, and the output is Heads;

2. Alice selects $a = 0$ with probability α, 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 β, 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 $\mathsf{Prob}[\text{Heads}] = \alpha\beta + (1-\alpha)(1-\beta) = 1 - \alpha - \beta + 2\alpha\beta$. If both players are dishonest, the protocol will not necessarily function correctly. If one of the parties is honest, so α or $\beta = 1/2$, then $\mathsf{Prob}[\text{Heads}] = 1/2$. Based on the above, we may be able to argue that the protocol is secure against malicious behavior in the following sense: no matter the behavior of a malicious party, assuming that the protocol is executed in its entirety, a uniformly distributed probability will be guaranteed to an honest party.

1.2 Overview of Cryptography

The previous example illustrates the research process of modern cryptography which can be epitomized as follows.

1. Identify important problems in need of *cryptographic* solutions. These are typically problems of trust between two or more parties. As a rule of thumb, if the problem can be solved by introducing a trusted "third" party that is connected to all the participants then the problem can be solved cryptographically. For instance, we will see that the coin tossing is an important problem that accepts a cryptographic solution and has numerous applications in constructing secure systems. Observe that it can be easily solved by employing a trusted third party that will flip a coin and announce it to both Alice and Bob.
2. Formally defining security and correctness for all involved parties. This some times is called the security model or threat model. It entails a formal definition of what the adversary is allowed to do and what is the objective it has.
3. Specify what *resources* are available to the parties that are engaged in the protocol. For instance in the solution above for coin flipping it was assumed that Alice and Bob both have a coin and they can flip to produce local randomness.
4. Design a candidate solution that is in the form of a protocol or algorithm and syntactically is consistent with the problem.
5. Determine a set of assumptions that are needed as preconditions for the solution to satisfy the security model. In the above description we made two assumptions that were informally stated about the function f, the binding and hiding properties.
6. Finally, provide a proof of security and correctness so as to convince that the system satisfies the security and correctness specifications as defined in the formal security model.

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 proofs 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.

The provable-security paradigm typically entails 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 is either impossible in the model or it can be transformed into an algorithm solving a "computationally hard" problem, i.e., a problem that we assume infeasible to be solved.

The second item points to an area of interest for us 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 many proofs 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, while NP problems may be hard only in the worst-case.

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 α that solves A with oracle access to B, written α^B. We can appropriately define a *pre-order* $\leq$[1] over all problems so $A \leq B$ if and only if there is an algorithm α where α^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 (that we will define in the sequel), 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. [2]

2.1 Algebra and Number Theory

2.1.1 Groups

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

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

Formally, a group is denoted by an ordered pair $(\mathbb{G}, *)$. We will write $\mathbb{G}$ when $*$ is clear from the context.

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

[1] A ***pre-order*** is a reflexive, transitive relation.
[2] For more mathematical review, see [1].

Proof. For the first part, let e, f be two identity elements in $\mathbb{G}$. Then we write:

$$\begin{aligned} e * f &= e * f \\ e &= e * f, && \text{because } f \text{ is an identity element} \\ e &= f, && \text{because } e \text{ is an identity element} \end{aligned}$$

For the second part, let a^{-1} be an inverse of a (at least one such inverse must exist because $\mathbb{G}$ is a group), and b be any element such that $a * b = 1$ (the equivalent case where $b * a = 1$ is left to the reader). We have:

$$\begin{aligned} a * b &= 1 \\ a^{-1} * a * b &= a^{-1} * 1 \quad \text{(we multiply both sides by it from the left)} \\ b &= a^{-1} \end{aligned}$$

In the above, we have also (indirectly) demonstrated that if $a * b = 1$ then it must also hold that $b * a = 1$ even if $*$ is not commutative in general. ∎

Definition 2.1.2. A group $\mathbb{G}$ is called ***Abelian*** (or commutative) if for all $g, h \in G$, $g * h = h * g$.

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

Definition 2.1.4. For any group $\mathbb{G}$ and any nonempty subset $\mathbb{H}$ of $\mathbb{G}$, we say that $\mathbb{H}$ is a subgroup of $\mathbb{G}$ if $\mathbb{H}$ is closed under $*$ and also closed under inversion.

Definition 2.1.5. For a group $(\mathbb{G}, *)$ and $g \in \mathbb{G}$, define the ***order of*** g to be the smallest positive integer i such that $g^i = e$, or equivalently, $\underbrace{g * g * \cdots * g}_{i \text{ times}} = e$. We denote the order of g by $\mathsf{ord}(g)$.

Definition 2.1.6. For any $g \in \mathbb{G}$ we define $\langle g \rangle$ to be the set $\{g^i : i \in \mathbb{Z}\}$.

We can show that $\langle g \rangle$ is a subgroup of $\mathbb{G}$. It is also clear that it has order $\mathsf{ord}(g)$.

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

Lagrange's theorem is extremely powerful, but for the time being we will point out two useful corollaries for finite groups:

Corollary 2.1.2.1. *For any $g \in \mathbb{G}$, $g^{\mathsf{ord}(\mathbb{G})} = 1$*

Corollary 2.1.2.2. *For any $g \in \mathbb{G}$, $g^{\mathsf{ord}(\mathbb{G})-1} = g^{-1}$*

The proofs of both corollaries are left as exercises. In practice, the two corollaries give us a way to cancel out or invert an element g even if we do not know its order.

Lagrange's theorem also gives us a way to determine an the order of an element g, if the group order is known: we only need to test each divisor of $\mathbb{G}$ as a potential order of g. The least divisor d of $\mathbb{G}$ such that $g^d = 1$ is g's order.

Definition 2.1.7. If there is some element $g \in \mathbb{G}$ such that $\mathsf{ord}(g) = \#\mathbb{G}$, then g generates $\mathbb{G}$ and we call $\mathbb{G}$ a ***cyclic group***. We write $\mathbb{G} = \langle g \rangle$.

Proposition 2.1.1. *Cyclic groups are Abelian.*

2.1.2 Elementary Number Theory

Definition 2.1.8. We say that a divides b iff there exists c such that $b = ac$. We denote this by $a|b$.

Definition 2.1.9. A positive integer p is prime iff it is only divisible by 1 and itself.

Definition 2.1.10. We define $\gcd(a, b)$ to be the largest number d such that d divides both a and b. If it is clear from the context, we may write $\gcd(a, b)$ as (a, b). When $(a, b) = 1$ we say that a and b are ***coprime***.

Lemma 2.1.1 (Bézout). *For any a, b there exist x, y such that $ax + by = \gcd(a, b)$*

Proof. Let $\delta = \gcd(a, b)$. Let d be the minimum positive element of the set $S := \{ax + by | x, y \in \mathbb{Z}\}$ (the set is non-empy as it contains e.g. a). We will show that $\delta = d$.

As d is of the form $ax + by$ and δ divides both a and b it is clear $\delta|d$, thus $\delta \leq d$.

We will now show that d divides a. We use euclidean division to write $a = kd + r$ where $0 \leq r < d$. Equivalently, $r = a - kd$ which implies $r \in S$. As $r < d$, this is turn implies r must be zero, as d is the least positive element of S. Thus, d divides a. We repeat the argument to show d also divides b. Thus d is a common divisor of a, b and must be $d \leq \delta$ as δ is the gcd.

We have $\delta \leq d$ and $d \leq \delta$, so it must be that $d = \delta$. ∎

We can use this to easily prove:

Lemma 2.1.2 (Euclid's Lemma). *If p divides $a \cdot b$, then p must divide at least one of a, b.*

2.1.3 The multiplicative group $\mathbb{Z}_p^*$

Definition 2.1.11. We denote ***congruence relationships*** over the integers by $a \equiv b \mod n$ if and only if $n \mid (a - b)$.

Definition 2.1.12. We denote the equivalence class $\bar{a}_n$ of a number $a \in \mathbb{Z}$ modulo n as : $\bar{a}_n := \{b : b - a \equiv 0 \mod n\}$. Given a class a_n, we define the *representative* of the class as the least positive member of the class.

Definition 2.1.13. We denote the set of integers modulo n as the set of equivalence classes modulo n: $\mathbb{Z}_n := \{\bar{a}_n\}$.

When it is clear from the context we will use equivalence classes, their representatives and their members interchangeably.

Example. Consider $\mathbb{Z}_5^* = \mathbb{Z}_5 - \{0\}$. This is a cyclic group under multiplication modulo 5.

Proof. Closure under multiplication and associativity hold by inspection. The identity element is 1, so it remains to show that each element $g \in \mathbb{Z}_5^*$ is invertible. Since 5 is prime, (g,5)=1 and by Lemma 2.1.1 there exist x, y such that $5x + gy = 1$ in the integers. This in turn implies that $gy - 1 = 5x$ i.e.

$$gy \equiv 1 \mod 5$$

∎

Let us now investigate the structure of $\mathbb{Z}_5^*$. Our goal is to find $g \in \mathbb{Z}_5^*$ such that $\mathsf{ord}(g) = \#\mathbb{Z}_5^* = 4$ and therefore $\langle g \rangle = \mathbb{Z}_5^*$. Clearly $\langle 1 \rangle \neq \mathbb{Z}_5^*$, so let us try 2:

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

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

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

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

A useful tool in $\mathbb{Z}_p^*$, is Fermat's little theorem, which can be easily proved using Lagrange's Theorem (2.1.2).

Theorem 2.1.3 (Fermat's little theorem). *For any prime p and any integer a: $a^p \equiv a \bmod p$.*

When a is non-zero $\bmod p$ we can also write this as $a^{p-1} \equiv 1 \bmod p$.

Extended Euclidean Algorithm We end this section with a reminder of integer division from school as well as it's extended version. We know that for any integers a, b there exist **unique** r, q such that $a = qb + r$ where $0 \leq r < |b|$. We say that b is the divisor, q is the quotient and r is the remainder.

The standard Euclidean Algorithm calculates (a, b) as follows: We divide $a := r_0$ by $b := r_1$ and obtain remainder r_2. We then divide b by r_2 to obtain r_3. We proceed by dividing r_i by r_{i+1} to derive r_{i+2}. We stop when we one of the r_i values is 0, and output r_{i-1}, which is the gcd of a, b.

The Extended Euclidean Algorithm not only calculates (a, b) but also a pair of x, y values such that $ax + by = \gcd(a, b)$. Initially, it operates in the same way as the standard version, with the addition of keeping the quotient values q_i from every division performed. When the original algorithm finishes, we know the gcd r_i^* as well as it's derivation in terms of r_{i^*-1} and r_{i^*-2}. We also know how to express r_{i^*-1} in terms of r_{i^*-2}, r_{i^*-3}. By iterating over i we are able to express r_i^* as a function of (a, b).

Let us divide 39 by 7:

$39 = 5 \cdot 7 + 4$

$7 = 1 \cdot 4 + 3$ We divide the previous divisor ($r_1 = 7$) by the previous remainder ($r_2 = 4$).

$4 = 1 \cdot 3 + 1$ We divide the previous divisor (4) by the previous remainder (3).

$3 = 3 \cdot 1 + \mathbf{0}$ Remainder r_5 is 0, we stop; the gcd is the previous remainder (1).

We now move to the extended part

$1 = 4 - 1 \cdot 3$ Start with the last non-zero remainder $r_4 = 1$.

$1 = 4 - 1 \cdot (7 - 1 \cdot 4)$ Replace $r_3 = 3$.

$1 = 2 \cdot 4 - 7$ Rewrite for clarity.

$1 = 2 \cdot (39 - 5 \cdot 7) - 7$ Replace $r_2 = 3$.

$1 = 2 \cdot 39 - 11 \cdot 7$ Rewrite.

Corollary 2.1.3.1. *We can directly calculate inverses in $\mathbb{Z}_p^*$.* Compared to Cor. 2.1.2.2, the Extended Euclidean might seem more involved, but is actually much more efficient, as the group order tends to be large.

The group $\mathbb{Z}_n^*$ for $n = pq$

In the previous section, we created $\mathbb{Z}_p^*$ from $\mathbb{Z}_p$ by simply excluding 0 since it has no multiplicative inverse. When working modulo n where n is composite, it is simple to see that the only elements with inverses are those co-prime to n.

Theorem 2.1.4. *A number a is invertible modulo n iff $(a, n) = 1$*

Proof. The first direction is straightforward: if (a,n)=1, we can use the Extended Euclidean to calculate the inverse.

For the second direction, assume $ab \equiv 1 \mod n$ i.e. $ab - 1 = kn$ for some k. This implies $1 = ab - kn$. The gcd of a, n divides a and also ab. It also divides k and also kn. Thus it must divide the difference $ab - kn$, which implies the gcd is 1, so a, n are co-prime. ■

Theorem 2.1.5. *The invertible elements of $\mathbb{Z}_n$ under multiplication form an Abelian group.*

Proof. The existence of inverses is true by assumption. We check that 1 is invertible and that it is also the neutral element. Associativity and commutability hold by inspection. Closure holds by observing that $(ab)^{-1} = a^{-1}b^{-1}$ thus, the product of two invertible elements is also invertible. ■

Note. $\mathbb{Z}_n^*$ in not cyclic in the general case. In the special case where $n = pq$ for different non-trivial primes p, q, it is explicitly not cyclic.

For convenience, we will state a very useful result that can be proved either directly, or using the Chinese Remainder Theorem (2.1.6), which we will be visiting later.

Corollary 2.1.5.1. *The number of invertible elements in $\mathbb{Z}_n^*$ for $n = p \cdot q$ with p, q different primes is $\phi(n) = (p-1) \cdot (q-1)$.*

Rings and Fields

Definition 2.1.14. 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 2.1.15. 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 2.1.16. Let p be a prime. Then $\mathbb{Z}_p$ is a ***finite field***, denoted $\mathbb{F}_p$.

Chinese Remainder Theorem

Theorem 2.1.6 (Chinese Remainder Theorem). *Let $m_1, \ldots, m_k$ be pairwise relatively prime positive integers and let $c_1, \ldots, c_k$ be arbitrary integers. Then there exists an integer x such that*

$$x \equiv c_i \bmod m_i$$

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

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

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

as desired. ∎

Remark. The Chinese Remainder Theorem implies the group isomorphism

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

given by $a \bmod n \mapsto (a \bmod p_1^{e_1}, \ldots, a \bmod p_m^{e_m})$, where $n = p_1^{e_1} \cdots p_m^{e_m}$ 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

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

We compute $M = 3 \cdot 5 \cdot 7 = 105$, and $m_1' = 35, m_2' = 21, m_3' = 15$. Computing inverses now, we have

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

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

2.2 Discrete Probability

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

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

The set $[\mathbf{D}]$ is called the ***support*** of **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 nth power, we obtain the upper bound $(1-p)^n \leq e^{-pn}$. Then

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

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:

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

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

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

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 $\mathsf{Prob}_{\mathbf{D}}[\text{Collision}] \geq 1/2$, 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 $[\mathbf{D}] = \{0, 1, \ldots, n\}$ 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:

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

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

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

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

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

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

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

The following theorem is useful for computing conditional probabilities.

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

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

Moreover, if $\mathbf{D}_1, \ldots, \mathbf{D}_n$ *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* $\mathbf{B}$*,*

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

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

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

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 $\mathsf{Prob}_{\mathbf{D}}[\mathbf{S}] = 1\%$, the chances of a successful test are $\mathsf{Prob}_{\mathbf{D}}[\mathbf{T} \mid \mathbf{S}] = 99\%$, and the probability of an inaccurate test is $\mathsf{Prob}_{\mathbf{D}}[\mathbf{T} \mid \overline{\mathbf{S}}] = 5\%$. 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

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

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

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

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

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

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

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

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

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

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 $t > 0$,*

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

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

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

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

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

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

Here μ 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

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

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

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

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

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

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

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

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

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

This implies that if the oracle has bias α, 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 ε, 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*** Δ by

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

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.
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 ε***-close*** if $\Delta[X,Y] \leq \varepsilon$.

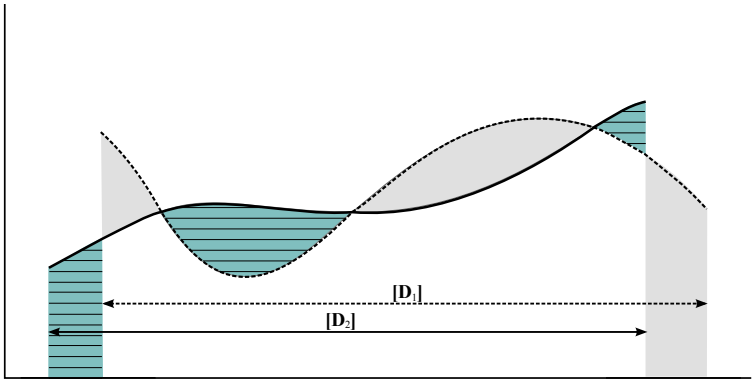


Figure 1: Two probability distributions over different support sets $[\mathbf{D}_1]$ and $[\mathbf{D}_2]$. The shaded regions distinguish the statistical distance between the random variables.

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

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

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

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

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 $\Delta[X,Y] \to 0$ 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 Statistical Tests

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*

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

*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 $\mathcal{A}$,

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

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 is 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

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

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

Now we observe that

$$\begin{aligned} N_1 + N_2 &= \sum_{a\in A_n} |\mathsf{Prob}_{\mathbf{D}_1}[a] - \mathsf{Prob}_{\mathbf{D}_2}[a]| + \sum_{a\in\overline{A_n}} |\mathsf{Prob}_{\mathbf{D}_1}[a] - \mathsf{Prob}_{\mathbf{D}_2}[a]| \\ &= \sum_{a\in\mathbf{D}_n} |\mathsf{Prob}_{\mathbf{D}_1}[a] - \mathsf{Prob}_{\mathbf{D}_2}[a]| \\ &= 2\Delta[X,Y]. \end{aligned}$$

Due to $\Delta_{\mathcal{A}}[X,Y] = \Delta_{\overline{\mathcal{A}}}[X,Y]$ and the fact that $\Delta_{\mathcal{A}}[X,Y] + \Delta_{\overline{\mathcal{A}}}[X,Y] \leq 2\cdot\Delta[X,Y]$ the result follows.

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

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

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

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

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 $\mathsf{Prob}_{\mathbf{D}_1}[u] \geq \mathsf{Prob}_{\mathbf{D}_2}[u]$, which we have already seen is exactly the statistical distance.

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

$\mathtt{b}_1\mathtt{b}_0$	$\mathbf{D}_1$	$\mathbf{D}_2$
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

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

Take a set of statistical tests $\mathcal{A}_1, \ldots, \mathcal{A}_5$ that distinguish the previous probability distributions. Suppose we are given two bits $\mathtt{b}_0$ and $\mathtt{b}_1$. Test $\mathcal{A}_1$ outputs $\mathtt{b}_1$. 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 $\mathtt{b}_0$, so then $\Delta_{\mathcal{A}_2}[X,Y] = |(0.25+0.25) - (0.25+0.25)| = 0$. If Test $\mathcal{A}_3$ outputs $\mathtt{b}_0 + \mathtt{b}_1 \bmod 2$, also denoted by the exclusive-or operator $\mathtt{b}_0 \oplus \mathtt{b}_1$, its statistical distance is given by $\Delta_{\mathcal{A}_3}[X,Y] = |0.25 + (0.25+\varepsilon) - (0.25+0.25)| = \varepsilon$. Test $\mathcal{A}_4$ outputs $\mathtt{b}_0 \vee \mathtt{b}_1$, so $\Delta_{\mathcal{A}_4}[X,Y] = |0.25 + (0.25+\varepsilon) + 0.25 - (0.25+0.25+0.25)| = \varepsilon$. And finally, if Test $\mathcal{A}_5$ outputs $\mathtt{b}_0 \wedge \mathtt{b}_1$, 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 $x \xleftarrow{\mathtt{r}} \{0,1\}$ 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

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

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, ··· , x_{n-1} ←r {0,1}
3:  if Σ_{i=0}^{n-1} 2^i x_i ≥ 2^{n-1}
4:     then output 1
5:     else output 0
```

Since 1 is a possible output,

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

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

```
1:  Input 1^n
2:  repeat n times
3:      x <-r {0,1}
4:      if x = 1, output 1 and halt
5:  output Fail
```

Then we have the following probabilities

$$\mathsf{Prob}[\mathcal{A}_2(1^n) = \mathtt{Fail}] = \frac{1}{2^n}$$
$$\mathsf{Prob}[\mathcal{A}_2(1^n) = 1] = 1 - \frac{1}{2^n}.$$

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 $\{0, 1, 2, \ldots, A-1\}$.

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 := ⌈log_2 A⌉
2:  choose: x_0, x_1, ..., x_{n-1} <-r {0,1}
3:  y := Σ_{i=0}^{n-1} 2^i x_i
4:  output y mod A
```

Sampler 2:

```
1:  choose: x_0, x_1, ..., x_{A-1} <-r {0,1}
2:  y := Σ_{i=0}^{A-1} x_i
3:  output y
```

Sampler 3:

```
1:  n := ⌈log_2 A⌉
2:  repeat
3:  choose: x_0, x_1, ..., x_{n-1} <-r {0,1}
4:  y := Σ_{i=0}^{n-1} 2^i x_i
5:  if y < A output y and halt
6:  else repeat
```

3 Constructing Commitment Schemes

We now turn our attention to the construction of the first cryptographic primitive that we mentioned in the context of coin flipping.

3.1 Syntax of a commitment scheme

Let λ be the security parameter; we can think of this value as the key length. Abstractly the commitment scheme can be divided in two stages: The commit stage and the open stage. Alice is the committer (or sender) and Bob is the receiver (or verifier). The commitment scheme may or may not be parameterized by a public parameter produced by an algorithm Gen. We will focus on "non-interactive" commitment schemes (i.e., schemes that require no interaction beyond a single message). The algorithms involved in a commitment scheme are given below.

Parameter generation . A trusted execution of $\mathsf{Param}(1^\lambda)$ is ensured and the output b is provided to both parties. Note that in case Param is deterministic a trusted execution of b can be trivially ensured by both parties by essentially repeating the computation whenever b is needed.

Commit Stage 1. Alice selects her message M and commits to it by calculating $(r, c) \leftarrow \mathsf{Commit}(b, M)$

2. Alice sends c to Bob

Open Stage 4. Alice sends the "decommitment" information r and the message M to Bob

5. Bob runs the verification algorithm to ensure the opening of Alice is appropriate.

$$\mathsf{Verify}(b, c, r, M) = \begin{cases} 1 & accept \\ 0 & reject \end{cases}$$

3.2 Security Properties

We next define the two security properties of the commitment scheme. We consider first the case that the parameters are generated by a trusted party.

Correctness: For every message M we require that: If $b \leftarrow \mathsf{Param}(1^\lambda)$ and $(r, c) \leftarrow \mathsf{Commit}(b, M)$, then $\mathsf{Verify}(b, c, r, M) = 1$, where b, c, r are chosen randomly.

Binding: Intuitively, this property requires that Alice should not be in position to change her message after sending her commitment. More formally let A be an algorithm that can commit two different messages M_1, M_2 to the same commitment c, that is

Algorithm 1 $bindattack^A(1^\lambda)$

1: Let $b \leftarrow \mathsf{Param}(1^\lambda)$
2: $(c, r_1, M_1, r_2, M_2) \leftarrow A(b)$
3: **if** $\mathsf{Verify}(b, c, r_1, M_1) = 1$ and $\mathsf{Verify}(b, c, r_2, M_2) = 1$ and $M_1 \neq M_2$ **then**
4: **return** 1
5: **else**
6: **return** 0
7: **end if**

We require that for every PPT A the following holds:

$$\mathsf{Prob}[bindattack^A(1^\lambda) = 1] = \mathsf{negl}(1^\lambda)$$

Strengthening of the above attack allows the adversary to specify the commitment parameter b. In this case we modify lines 1-2 above so that $(b, c, r_1, M_1, r_2, M_2) \leftarrow A(1^\lambda)$. In this case we will say that we have a non-interactive commitment scheme that satisfies hiding with adversarial parameters.

Hiding: The secrecy of Alice's message is preserved, meaning that Bob cannot extract any information about Alice's message

Algorithm 2 $hidingattack^{B}(1^{\lambda})$

1: $b \leftarrow \mathsf{Param}(1^{\lambda})$
2: Let $(aux, M_0, M_1) \leftarrow B_1(1^{\lambda})$
3: $d \xleftarrow{R} \{0,1\}$
4: $(r, c) \leftarrow \mathsf{Commit}(b, M_d)$
5: $d^* \leftarrow B_2(c, aux)$
6: **if** $d^* = d$ and $M_0 \neq M_1$ **then**
7: **return** 1
8: **else**
9: **return** 0
10: **end if**

We require that for every pair of PPT algorithms $B = (B_1, B_2)$

$$\mathsf{Prob}[hidingattack^{B}(1^{\lambda}) = 1] \leq \frac{1}{2} + \mathsf{negl}(1^{\lambda})$$

As before strengthening of the above attack allows the adversary to specify the commitment parameter b. In this case we modify lines 1-2 above so that $(b, aux, M_0, M_1) \leftarrow B_1(1^{\lambda})$. In this case we will say that we have a non-interactive commitment scheme that satisfies binding with adversarial parameters.

3.2.1 Statistical and Perfect Security

In the above, we require that the adversaries A,B are computationally bound, i.e. their running time is polynomial. We can thus more accurately describe the security properties prescribed by the above definitions as *computationally hiding* and *computationally binding*.

We can strengthen either of these definitions to allow the adversary unlimited running time. In that case, the relevant security property is said to hold *statistically* rather than *computationally*. We can further strengthen such definitions by replacing the $negl(1^{\lambda})$ term with zero. We call this *perfect* security.

We note that the same scheme may achieve different levels of security over different properties. For example, the Pedersen commitment scheme we will describe in section 3.4 can be shown to be computationally binding and perfectly hiding.

3.3 The Discrete Logarithm Problem

Now that we have specified what a commitment scheme is, it is time to see a way to implement it. We will be describing Pedersen's protocol, which relies on the discrete logarithm assumption for security. Thus, before presenting the protocol's description, we will first define the Discrete Logarithm Problem.

As a reminder, the logarithm is the inverse of the exponentiation function, i.e if $y = g^x$, we say that x is the logarithm of y with respect to g, (or under base g). We may also write this as $x = \log_g y$.

3.3.1 Group Generators

Definition 3.3.1. A ***group generator*** GGen is a probabilistic algorithm that produces a description of a finite group $\mathbb{G}$ when given a length λ. At a minimum, the description contains a group element, the group operation, and a group membership test.

In practice, we will assume GGen will provide us with a generating element g, as well as its order. It is also common for the group $\mathbb{G}$ that is provided by GGen to be a supergroup of $\langle g \rangle$. This will be made clear in the context.

Example. Take $\mathbb{Z}_p^*$ to be our group for some prime p of length λ. GGen returns an element g of order q, where q is some function of λ and p. The group operation is multiplication modulo p, and if an integer is between 1 and $p-1$, it passes the group membership test. (Note that g only produces a subgroup of $\mathbb{Z}_p$, and that the membership test is with respect to $\mathbb{Z}_p^*$).

To implement such a generator, GGen on input 1^λ can calculate a random number p of the form $4k+3$ that has λ 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 $a \in \{2, \ldots, p-2\}$ and random and computes $a^{(p-1)/2} \bmod p$. If this value is 1 then it chooses another a. Otherwise it sets $g = a^2 \bmod p$. The output of the algorithm GGen are the values $(p, g, q = (p-1)/2)$.

3.3.2 The Discrete Logarithm problem with respect to GGen

Suppose now we have a group generator GGen, that produces the description of a finite group $\mathbb{G}$ and of a cyclic subgroup of order q within $\mathbb{G}$. We consider only the case that q is a prime number. We assume that the order of $\mathbb{G}$ may be known via the group description, but we will not be using it directly. Based on this we define the following algorithm that samples discrete-logarithm parameters.

Algorithm 3 $\mathrm{DLP}^{\mathsf{GGen}}(1^\lambda)$

1: $\langle \mathbb{G}, q, g \rangle \leftarrow \mathsf{GGen}(1^\lambda)$
2: $t \xleftarrow{R} \mathbb{Z}_q$
3: $h \leftarrow g^t$
4: **return** $\langle \mathbb{G}, q, g, h, t \rangle$

For an algorithm $\mathcal{A}$, we say that it solves the Discrete Log Problem, with respect to GGen, if

$$\mathsf{Prob}[\langle \mathbb{G}, q, g, h, t \rangle \leftarrow \mathrm{DLP}^{\mathsf{GGen}}(1^\lambda) : \mathcal{A}(\mathbb{G}, q, g, h) = t]$$

is not negligible. Based on the above we can now define the DLog assumption, according to which:

$$\forall\, \mathrm{PPT}\ \mathcal{A} :\ \mathsf{Prob}[\langle \mathbb{G}, q, g, h, t \rangle \leftarrow \mathrm{DLP}^{\mathsf{GGen}}(1^\lambda) : \mathcal{A}(\mathbb{G}, q, g, h) = t] = \mathsf{negl}(\lambda)$$

3.4 Pedersen Commitments

We are now ready to describe Pedersen's protocol, and prove it secure under the Discrete Logarithm Assumption w.r.t a group generator GGen.

1. Param will call $\mathsf{GGen}(1^\lambda)$ which outputs $(\mathbb{G}, q, g)$ such that $g \in \mathbb{G}$, $q = order(g)$ (that is $g^q = 1$), q is prime $|q| = \lambda$ bits. Also, let $h = g^t$, where $t \xleftarrow{R} \mathbb{Z}_q$. Param outputs $b = \langle \mathbb{G}, q, g, h \rangle$.

2. To commit to a message M using parameters b, Alice checks that q is prime, that $g, h \in \mathbb{G}$ and that $g^q = h^q = 1$. She selects $r \xleftarrow{R} \mathbb{Z}_q$ and her message $M \in \mathbb{Z}_q$. She commits to her message:
$$c = g^r h^M$$
and sends her commitment c, while keeping M, r. If any of the checks fails she outputs "Bad parameters."

3. At the Opening/Verification stage, Alice sends the revelation r and her message M and Bob verifies using the condition:
$$\text{if } c = g^r h^M \text{ then 1 else 0}$$

3.4.1 Proof of security

The aforementioned protocol is secure, i.e. the Binding and Hiding properties hold.

Binding Suppose that the Binding property does not hold. Then there exists an algorithm $\mathcal{A}$, that is successful in the binding attack described earlier. We will use this adversary, to construct an algorithm that breaks the discrete-logarithm assumption and therefore end up in a contradiction.

The fact that $\mathcal{A}$ can perform a successful binding attack, means that it can find two pairs (r_1, m_1) and (r_2, m_2) such that

$$\left.\begin{array}{rl} c = & g^{r_1} h^{m_1} \\ c = & g^{r_2} h^{m_2} \end{array}\right\} \Rightarrow g^{r_1} h^{m_1} = g^{r_2} h^{m_2}$$

which means that

$$g^{r_1 - r_2} = h^{m_2 - m_1}$$

Using the extended Euclidean Algorithm it is easy to find $k \in \mathbb{Z}_m$ such that $k(m_2 - m_1) = 1$, which means that

$$g^{(r_1 - r_2)k} = h$$

and

$$(r_1 - r_2)k = \log_g h$$

But this contradicts our initial DLog assumption and therefore we can conclude, that such an adversary does not exist and our protocol preserves the binding property.

Hiding We will now show that the hiding property holds even with adversarial parameters. For this let us define a function

$$f : \mathbb{Z}_q \times \mathbb{Z}_q \to \langle g \rangle \equiv \mathbb{Z}_q$$

such that

$$f(r, M) = c$$

The above defined function f is surjective and bijective[3] for a fixed[4] $M \in \mathbb{Z}_q$.
It suffices to show that for $r_1, r_2 \xleftarrow{R} \mathbb{Z}_q$ and two messages M_1, M_2 the following holds:

$$\Delta[f(r_1, M_1), f(r_2, M_2)] = 0$$

or equivalently that

$$\Delta[f(r, M), U] = 0,$$

where U is the uniform distribution on $\mathbb{Z}_q$, which would evidently mean that c does not provide any information about M.

Let $z \in \langle g \rangle$, i.e. $z = g^l$, where $l \in \mathbb{Z}_q$. Then we have that:

$$\begin{array}{rcl} \mathsf{Prob}[f(r, M) = z] & = & \mathsf{Prob}[g^r h^M = g^l] \\ & = & \mathsf{Prob}[g^r = g^l h^{-M}] \\ & = & \mathsf{Prob}[g^r = g^{l - M \cdot t}] \\ & = & \mathsf{Prob}[r = l - M \cdot t \mod q] \\ & = & \dfrac{1}{q} \end{array}$$

[3] Onto: Observe that $f(r, M) = g^r h^M = g^r (g^t)^M = g^{r + tM}$. So for $a \in \mathbb{Z}_q$ we have that there exists an r, such that $f(r, M) = a$
1-1: Suppose there exist r_1 and r_2 such that $f(r_1, M) = f(r_2, M)$ Then we have that $f(r_1, M) = f(r_2, M) \Rightarrow g^{r_1 + T} = g^{r_2 + T}$ for a $T = tM$
$\Rightarrow r_1 + T \equiv r_2 + T \mod m \Rightarrow r_1 \equiv r_2 \mod q$
Observe that we have used all the conditions, that Alice checks when she receives the parameters from Bob.

[4] Observe that if we allow M to vary, bijectivity breaks (take for example $(5, 0)$ and $(2, 1)$ in $\mathbb{Z}_7^*$ with $g = 3$)

Which proves that

$$\Delta[f(r, M), U] = 0,$$

The remaining of the proof now goes in two steps: first we modify line 4 of the $hidingattack^B(1^\lambda)$ so that c is selected at random from $\langle g \rangle$. For this modified attack game $\overline{hidingattack}^B(1^\lambda)$ it will hold that:

$$\mathsf{Prob}[hidingattack^B(1^\lambda) = 1] = \mathsf{Prob}[\overline{hidingattack}^B(1^\lambda) = 1]$$

Furthermore, if W_b is the event $d^* = d \wedge M_0 \neq M_1$, in the attack game $\overline{hidingattack}^B(1^\lambda)$ we have

$$\mathsf{Prob}[\overline{hidingattack}^B(1^\lambda) = 1] = \mathsf{Prob}[W_0 \mid d = 0]\mathsf{Prob}[d = 0] + \mathsf{Prob}[W_1 \mid d = 1]\mathsf{Prob}[d = 1]$$

We observe that the random variable d is independent from W_0 and W_1 and thus

$$\mathsf{Prob}[\overline{hidingattack}^B(1^\lambda) = 1] = (\mathsf{Prob}[W_0] + \mathsf{Prob}[W_1])\frac{1}{2} = \frac{1}{2}$$

This completes the proof.

3.4.2 Notes and pitfalls

We note that in order to achieve security against adversarial parameters, the checks performed by Alice are essential. As a concrete example, we will use the very natural example of $\mathbb{Z}_p^*$ as the basis for our group.

As $p = 2k+1$ for most primes p, $\mathbb{G}$ will be a group of order $2k$ and thus our order q subgroup is non-trivial (i.e. not the same as $\mathbb{G}$ itself).

Suppose the adversary chooses parameters such that $order(h) = 2q$. For example, $h = \zeta g^t$ and ζ such that[5] $order(\zeta) = 2$ in $\mathbb{Z}_p^*$), also note that $\zeta^q = \zeta$ as q is odd. Then, after receiving Alice's commitment c he just needs to do the following in order to compute one bit of M :

$$c^q = (g^r)^q(h^M)^q = (h^q)^M = (\zeta^q)^M(g^q)^{tM} = \zeta^M = \begin{cases} 1, & M \text{ even} \\ \zeta, & M \text{ odd} \end{cases}$$

This would constitute a hiding attack against a protocol that does not check that the order of h is indeed q. The above shows that the test $h^q = 1$ is essential for the hiding property with adversarial parameters.

What about the other tests that Alice performs? Can you find attacks if they are omitted?

We can also show that binding breaks down if Alice is the one generating the parameters. As a first indication of that, we point out that our proof is no longer meaningful. Recall that we show that an adversary who can break the binding property can be used to compute a discrete logarithm by having the experiment control the commitment parameters b. But, if Alice is the one generating the parameters, this no longer works. Essentially, we are letting Alice choose a discrete log that she already knows.

More concretely, if we know that $h = g^x$ we may simply choose (r, M) and $(r - x, M + 1)$ as the two openings of $c = g^r h^M$. Details are left as an exercise.

3.5 Choosing a Group for the DLP

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.

[5] E.g. $\zeta \equiv -1 \mod p$

Example (Silver–Pohlig–Hellman Algorithm). 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 $\mathbb{G}_i$ of order q_i.[6] Define the group homomorphism $f_i \colon \mathbb{Z}_p^* \longrightarrow \mathbb{G}_i$ 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 $y = g^x \bmod p$. Raising both sides to the $p - 1/q_i$ power, we have $y^{p-1/q_i} \equiv (g^{p-1/q_i})^x \equiv g_i^{x \bmod q_i} \bmod p$ 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 $x_i \equiv x \bmod q_i$. 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 residues of $\mathbb{Z}_p^*$.

Definition 3.5.1. The ***quadratic residues*** of $\mathbb{G}$ is the subgroup of all $y \in \mathbb{G}$ such that there is an $x \in \mathbb{G}$ with $x^2 = y$.

When $\mathbb{G} = \mathbb{Z}_n^*$, we write the quadratic residues as $QR(n)$. In the particular case $\mathbb{G} = \mathbb{Z}_p^*$ for a prime p, $QR(p) = \langle g^2 \rangle$ for a generator g of $\mathbb{G}$. $QR(p)$ 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 3.5.1. *Consider some $a \in \mathbb{Z}$ and $p \equiv 3 \bmod 4$. It holds that $a^{\frac{p-1}{2}} = 1 \bmod p$ if and only if $a \in QR(p)$.*

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

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

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

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

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 $\pm a^{\frac{p+1}{4}} \bmod p$.

4 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 4.0.1. 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 $\mathcal{E} \colon \mathcal{K} \times \mathcal{M} \longrightarrow \mathcal{C}$
- An efficient decryption algorithm $\mathcal{D} \colon \mathcal{K} \times \mathcal{C} \longrightarrow \mathcal{M}$
- An efficient key generation algorithm $\mathcal{G} \colon \mathbb{N} \longrightarrow \mathcal{K}$

[6] The existence of such a subgroup is guaranteed by Cauchy's Theorem.

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

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

4.1 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 $\pi \colon \mathcal{M} \longrightarrow \mathcal{C}$ and the decryption algorithm performs the inverse substitution $\pi^{-1} \colon \mathcal{C} \longrightarrow \mathcal{M}$.

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: $\mathcal{E}((a, b), m) = am + b \bmod N$

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 $\{0, 1, \ldots, N-1\}$. 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”,

p	r	o	c	e	e	d
g	o	l	d	g	o	l
v	f	z	f	k	s	o

$\longleftrightarrow$

15	17	14	2	4	4	3
6	14	11	3	6	14	11
21	5	25	5	10	18	14

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 π. The decryption algorithm recovers the original message by applying the inverse permutation π^{-1}.

Example (Transposition Cipher).

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

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

4.2 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 $f\colon \{0,1\}^{48} \times \{0,1\}^{32} \longrightarrow \{0,1\}^{32}$ (we will formally define f later in the section) and 48-bit string keys $k_1, k_2, \ldots, k_{16}$ 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*** $F_k\colon \{0,1\}^{64} \longrightarrow \{0,1\}^{64}$ 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

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

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

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

Lemma 4.2.1. *Let $\mathcal{F}$ be the set of all functions $F\colon \{0,1\}^{64} \longrightarrow \{0,1\}^{64}$. It follows that for all $m > 0$ and $F_1, F_2, \ldots, F_m \in \mathcal{F}$,*

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

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 $F_1, F_2, \ldots, F_i \in \mathcal{F}$, 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 4.2.1. *The DES cryptosystem satisfies the correctness property.*

Proof. We prove this for a two-round DES scheme. Using Lemma 4.2.1, 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 4.2.1. ∎

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, $f\colon \{0,1\}^{48} \times \{0,1\}^{32} \longrightarrow \{0,1\}^{32}$ 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 $B_1, B_2, \ldots, B_8$, where each B_i is a 6-bit block.
4. Determine $[S_1(B_1), \ldots, S_8(B_8)]$, where $S_i\colon \{0,1\}^6 \longrightarrow \{0,1\}^4$ is a fixed substitution map for each i.
5. Apply a final fixed permutation.

The key schedule $k_1, k_2, \ldots, k_{16}$ 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 ith 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, may prominent cryptographers maintained that the NSA could break the DES encryption by brute force.

4.3 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,

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

The key is divided in a similar fashion,

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

Rijndael operates on the 4×4 matrix representations of the InputBlock and KeyBlock:

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

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 $\mathsf{Round}(State, RoundKey)$, where *State* is the 4×4 matrix returned by the previous transformation and *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 $B = \{0,1\}^8$,

$$\mathsf{Round}\colon B^{4\times 4} \times B^{4\times 4} \longrightarrow B^{4\times 4},$$

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

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

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

The final round, denoted $\mathsf{FinalRound}(State, RoundKey)$, 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 Round^{-1} and $\mathsf{FinalRound}^{-1}$.

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 ith 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 8×8 lookup table A. Let $s_{ij} \in \mathbb{F}_{2^8}$ be a 1-byte element from *State* for $0 \le i, j \le 3$. After completing the SubBytes step, the *newState* matrix is

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

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

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

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 4.3.1. $X^i \equiv X^{i \bmod 4} \bmod X^4 + 1$.

Lemma 4.3.2. $d(X) = d_3X^3 + d_2X^2 + d_1X + d_0$ *where*

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

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 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.

$\mathsf{Round}^{-1}(State, RoundKey)\{$
    $\mathsf{AddRoundKey}^{-1}(State, RoundKey)$;
    $\mathsf{MixColumns}^{-1}(State)$;
    $\mathsf{ShiftRows}^{-1}(State)$;
    $\mathsf{SubBytes}^{-1}(State)$;
$\}$

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.

5 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.

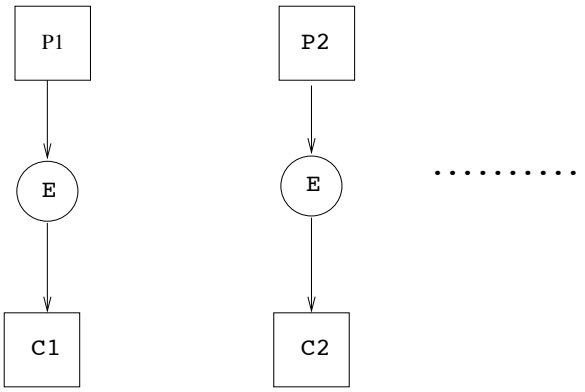


Figure 2: Electronic Codebook Mode

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 $P_1, P_2, \ldots$. 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.

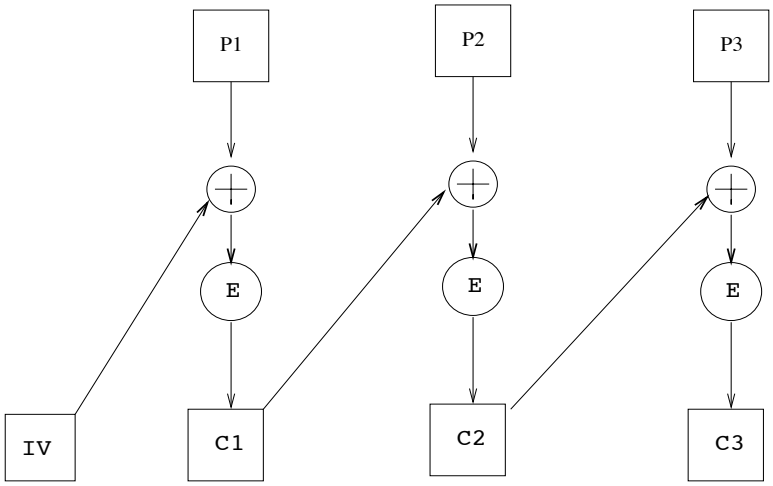


Figure 3: Cipher Block Chaining Mode

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.

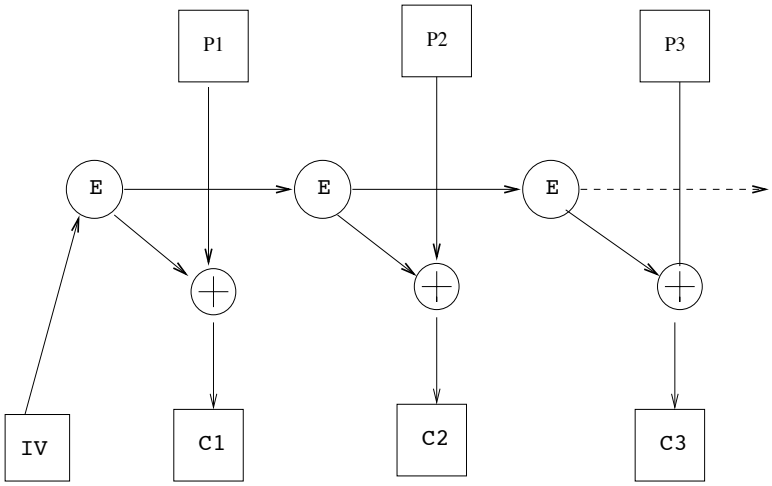


Figure 4: Output Feedback Mode

6 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.

6.1 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 $x \xleftarrow{r} \mathbb{Z}_q$ means that x is sampled according to the uniform over $\mathbb{Z}_q$. Observe that $y_B^{x_A} = y_A^{x_B} \bmod p$, so $k_A = k_B$ and both parties compute the same value in (the order q subgroup of) $\mathbb{G}$.

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.

6.2 Number-Theoretical Problems Related to DLP

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

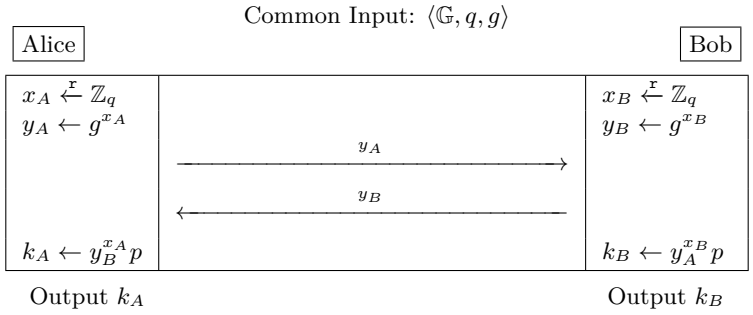


Figure 5: The Diffie-Hellman key exchange protocol, where g is a generator of a subgroup of $\mathbb{G}$ of order q.

reduce the security of the protocol to an appropriate number-theoretical assumption.

Definition 6.2.1. For a suitable cyclic group $\mathbb{G} = \langle g \rangle$, take $y \in \mathbb{G}$ of order q. The ***discrete logarithm problem*** (DL) is to find an integer $x \in \mathbb{Z}_q$ 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 6.2.2. Given a cyclic group $\mathbb{G} = \langle g \rangle$ of order q, g^a and g^b where $a, b \xleftarrow{r} \mathbb{Z}_q$, 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: CDH $\leq$ DL.

Lemma 6.2.1. *The computational Diffie-Hellman problem is no harder than the discrete logarithm problem.*

It is unknown if the converse holds.

Definition 6.2.3. The ***decisional Diffie-Hellman problem*** (DDH) is as follows: given a group $\mathbb{G} = \langle g \rangle$ of order q and g^a, g^b, g^c, where $a, b, c \xleftarrow{r} \mathbb{Z}_q$, decide if $c = ab$ or $c \xleftarrow{r} \mathbb{Z}_q$.

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, DDH $\leq$ CDH.

Lemma 6.2.2. *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.

6.3 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 6.3.1. The group generator 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:

$$\mathcal{D}_\lambda := \left\{ \langle \mathbb{G}, q, g \rangle \leftarrow \mathsf{GGen}(1^\lambda);\ a, b \xleftarrow{r} \mathbb{Z}_q \colon (\mathbb{G}, q, g, g^a, g^b, g^{ab}) \right\}$$
$$\mathcal{R}_\lambda := \left\{ \langle \mathbb{G}, q, g \rangle \leftarrow \mathsf{GGen}(1^\lambda);\ a, b, c \xleftarrow{r} \mathbb{Z}_q \colon (\mathbb{G}, q, g, g^a, g^b, g^c) \right\}$$

where $q = \mathsf{ord}(g)$.

Equivalently, if $\mathcal{A}$ is a statistical test bounded by probabilistic polynomial-time (PPT), then $\mathsf{Adv}^{\mathcal{A}}$, the ***advantage*** of $\mathcal{A}$ is negligible in λ, i.e. :

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

6.3.1 Statistical distance for DDH

It is of note that for the statistical distance $\Delta[\mathcal{D}_\lambda, \mathcal{R}_\lambda]$ we have $\Delta[\mathcal{D}_\lambda, \mathcal{R}_\lambda] \geq 1 - 2^{-\lambda}$ which is close to 1. In order to show why this result derives, first we consider that the number of possible values of a random variable which follows the $\mathcal{D}_\lambda$ distribution is q^2, because this is the number of all different pairs (g^a, g^b). On the other hand, the number of possible values of a random variable distributed according to $\mathcal{R}_\lambda$ is q^3. As a result, these values contain as a subset the values of the random variable distributed according to $\mathcal{D}_\lambda$. Now, we calculate the statistical distance as follows:

$$\begin{aligned} \Delta[\mathcal{D}_\lambda, \mathcal{R}_\lambda] &= \frac{1}{2}\left(q^2 \cdot \left| \frac{1}{q^2} - \frac{1}{q^3} \right| + (q^3 - q^2) \cdot \left| 0 - \frac{1}{q^3} \right| \right) \\ &= 1 - \frac{1}{q}. \end{aligned}$$

Due to q being a λ-bit integer, the result follows. This may seem paradoxical, as the result seems to imply that DDH is easy rather than hard. The key point is that the DDH assumption is *computational* i.e. even if the two distributions are very dissimilar, we believe it is hard to efficiently tell them apart.

6.4 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 $\mathtt{trans}_{A,B}(1^\lambda)$ 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 τ is denoted $key(\tau)$. 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[7] $\mathcal{A}$,

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

is a negligible function in λ. 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 $\log^2(\lambda)$.

[7] 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

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

for some negligible function $\mathsf{negl}(\lambda)$. 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

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

where α is non-negligible. Let $\mathcal{B}$ be a DDH distinguisher such that, given $\gamma = \langle \mathbb{G}, g, q, a, b, c \rangle$, $\mathcal{B}$ uses γ to form a transcript $\tau_\gamma = \langle \mathbb{G}, g, q, a, b \rangle$. $\mathcal{B}$ then simulates $\mathcal{A}$ on τ_γ 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 $\mathbb{G}$, let $\mathsf{Prob}[V(c) = 1] = \delta$.

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

2. If $\gamma \leftarrow \mathcal{R}_\lambda$, then $c \xleftarrow{\mathtt{r}} \mathbb{G}$ and

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

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

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

Looking at the DDH assumption,

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

Because α is non-negligible, $\mathcal{B}$ can break the DDH assumption when it has $\mathcal{A}$ and $\delta = 1/2$, which is what we wanted. However, this fortunate result relies critically on the value of δ.

When $\delta \neq 1/2$, 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

$$\underset{\tau \leftarrow \mathtt{trans}_{A,B}(1^\lambda)}{\mathsf{Prob}}[V(key(\tau)) = 1] = \delta,$$

where $\mathtt{trans}_{A,B}(1^\lambda)$ is the distribution of the transcripts of the interactions between two players A and B.

Definition 6.4.1. We say that a key exchange protocol is secure under the Security Model 3, if for any PPT adversary $\mathcal{A}$ and any predicate V such that $\underset{\tau \leftarrow \mathtt{trans}_{A,B}(1^\lambda)}{\mathsf{Prob}}[V(key(\tau)) = 1] = \delta$, it holds that

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

We continue by proving that Diffie-Hellman key exchange protocol is secure against passive adversaries under this security model. Namely, we will prove the following theorem.

Theorem 6.4.1. *If the DDH assumption holds, the Diffie-Hellman key exchange protocol is secure against passive adversaries under Security Model 3.*

Proof. Assume there exists a PPT adversary $\mathcal{A}$ and predicate V such that

$$\underset{\tau \leftarrow \mathtt{trans}_{A,B}(1^\lambda)}{\mathsf{Prob}}[\mathcal{A}(\tau) = V(key(\tau))] \geq \max\{\delta, 1-\delta\} + \alpha, \tag{2}$$

for a non-negligible α. We will construct a PPT distinguisher $\mathcal{B}$ that breaks the DDH assumption with non-negligible probability.

Let $\mathcal{B}$ be a PPT DDH distinguisher that on input $\gamma = \langle \mathbb{G}, q, g, a, b, c\rangle$ invokes $\mathcal{A}$ on input $\tau_\gamma = \langle G, q, g, a, b\rangle$ and receives its output s. If $V(c) = s$, then $\mathcal{B}$ outputs 1 otherwise it outputs 0. We will show that if (2) holds then

$$\left| \underset{\gamma \leftarrow D_\lambda}{\mathsf{Prob}}[\mathcal{B}(\gamma) = 1] - \underset{\gamma \leftarrow \mathcal{R}_\lambda}{\mathsf{Prob}}[\mathcal{B}(\gamma) = 1] \right| \geq \beta,$$

for β non-negligible (i.e $\mathcal{B}$ breaks DDH).

Behavior on DDH tuples: For $\gamma \leftarrow \mathcal{D}_\lambda$, we have that

$$\begin{aligned}
\underset{\gamma \leftarrow D_\lambda}{\mathsf{Prob}}[\mathcal{B}(\gamma) = 1] &= \underset{\gamma \leftarrow D_\lambda}{\mathsf{Prob}}[\mathcal{A}(\tau_\gamma) = V(c)] \\
&= \underset{\gamma \leftarrow D_\lambda}{\mathsf{Prob}}[\mathcal{A}(\tau_\gamma) = V(key(\tau_\gamma))] \\
&\geq \max\{\delta, 1-\delta\} + \alpha.
\end{aligned} \tag{3}$$

Behavior on random tuples: When $\gamma \leftarrow \mathcal{R}_\lambda$, it holds that

$$\begin{aligned}
\underset{\gamma \leftarrow \mathcal{R}_\lambda}{\mathsf{Prob}}[\mathcal{B}(\gamma) = 1] &= \underset{\gamma \leftarrow \mathcal{R}_\lambda}{\mathsf{Prob}}[\mathcal{B}(\gamma) = 1 | V(c) = 1] \underset{\gamma \leftarrow \mathcal{R}_\lambda}{\mathsf{Prob}}[V(c) = 1] + \underset{\gamma \leftarrow \mathcal{R}_\lambda}{\mathsf{Prob}}[\mathcal{B}(\gamma) = 1 | V(c) = 0] \underset{\gamma \leftarrow \mathcal{R}_\lambda}{\mathsf{Prob}}[V(c) = 0] \\
&= \mathsf{Prob}[\mathcal{A}(\tau_\gamma) = 1] \cdot \mathsf{Prob}[V(c) = 1] + \mathsf{Prob}[\mathcal{A}(\tau_\gamma) = 0] \cdot \mathsf{Prob}[V(c) = 0],
\end{aligned}$$

because c is a random element and therefore $\mathcal{A}$'s output is independent from the value $V(c)$. Let $\delta' = \underset{\gamma \leftarrow \mathcal{R}_\lambda}{\mathsf{Prob}}[V(c) = 1]$. Then, we have that

$$\begin{aligned}
\underset{\gamma \leftarrow \mathcal{R}_\lambda}{\mathsf{Prob}}[\mathcal{B}(\gamma) = 1] &= \mathsf{Prob}[\mathcal{A}(\tau_\gamma) = 1]\delta' + \mathsf{Prob}[\mathcal{A}(\tau_\gamma) = 0](1-\delta') \\
&\leq \mathsf{Prob}[\mathcal{A}(\tau_\gamma) = 1](\max\{\delta', 1-\delta'\}) + \mathsf{Prob}[\mathcal{A}(\tau_\gamma) = 0](\max\{\delta', 1-\delta'\}) \\
&= \max\{\delta', 1-\delta'\}\left(\mathsf{Prob}[\mathcal{A}(\tau_\gamma) = 1] + \mathsf{Prob}[\mathcal{A}(\tau_\gamma) = 0]\right) \\
&= \max\{\delta', 1-\delta'\}.
\end{aligned} \tag{4}$$

We assert that $|\delta - \delta'| \leq \frac{q-1}{q^2}$. We postpone the proof for readability. By the assertion, it holds that

$$|\delta - \delta'| \leq \frac{q-1}{q^2} \Leftrightarrow |(1-\delta) - (1-\delta')| \leq \frac{q-1}{q^2},$$

So, we have that

$$\delta' \leq \delta + \frac{q-1}{q^2} \text{ and } 1 - \delta' \leq 1 - \delta + \frac{q-1}{q^2}$$

which means that

$$\delta' \leq \max\{\delta, 1-\delta\} + \frac{q-1}{q^2} \text{ and } 1-\delta' \leq \max\{1-\delta\} + \frac{q-1}{q^2}$$

Consequently it holds that

$$\max\{\delta', 1-\delta'\} \leq \max\{\delta, 1-\delta\} + \frac{q-1}{q^2}. \tag{5}$$

Now, from (4), (5), we have

$$\underset{\gamma \leftarrow \mathcal{R}_\lambda}{\mathsf{Prob}}[\mathcal{B}(\gamma)=1] \leq \max\{\delta, 1-\delta\} + \frac{q-1}{q^2}. \tag{6}$$

Difference of the two behaviors Finally, from (3), (6)

$$\begin{aligned}\Big|\underset{\gamma \leftarrow \mathcal{D}_\lambda}{\mathsf{Prob}}[\mathcal{B}(\gamma)=1] - \underset{\gamma \leftarrow \mathcal{R}_\lambda}{\mathsf{Prob}}[\mathcal{B}(\gamma)=1]\Big| &\geq \underset{\gamma \leftarrow \mathcal{D}_\lambda}{\mathsf{Prob}}[\mathcal{B}(\gamma)=1] - \underset{\gamma \leftarrow \mathcal{R}_\lambda}{\mathsf{Prob}}[\mathcal{B}(\gamma)=1] \\ &\geq \big(\max\{\delta, 1-\delta\} + \alpha\big) - \big(\max\{\delta, 1-\delta\} + \frac{q-1}{q^2}\big) \\ &= \alpha - \frac{q-1}{q^2},\end{aligned}$$

which is non-negligible. By setting $\beta = \alpha - \frac{q-1}{q^2}$ the proof of security of the Diffie-Hellman protocol is completed.

■

Claim. *It holds that* $|\delta - \delta'| = \frac{q-1}{q^2} = \mathsf{negl}(\lambda)$.

Proof. Let $\mathcal{E}$ be a polynomial time algorithm that computes the predicate V. Then,

$$\delta = \underset{\gamma \leftarrow \mathcal{D}_\lambda}{\mathsf{Prob}}[V(c)=1] = \underset{x,y \xleftarrow{r} \mathbb{Z}_q}{\mathsf{Prob}}[\mathcal{E}(g^{xy})=1] \text{ and } \delta' = \underset{\gamma \leftarrow \mathcal{R}_\lambda}{\mathsf{Prob}}[V(c)=1] = \underset{t \xleftarrow{r} \mathbb{Z}_q}{\mathsf{Prob}}[\mathcal{E}(g^t)=1].$$

So,

$$\begin{aligned}\big|\delta - \delta'\big| = \Big|\underset{x,y \xleftarrow{r} \mathbb{Z}_q}{\mathsf{Prob}}[\mathcal{E}(g^{xy})=1] - \underset{t \xleftarrow{r} \mathbb{Z}_q}{\mathsf{Prob}}[\mathcal{E}(g^t)=1]\Big| &= \Delta_{\mathcal{E}}[\{x,y \xleftarrow{r} \mathbb{Z}_q : g^{xy}\}, \{t \xleftarrow{r} \mathbb{Z}_q : g^t\}] \\ &\leq \Delta[\{x,y \xleftarrow{r} \mathbb{Z}_q : g^{xy}\}, \{t \xleftarrow{r} \mathbb{Z}_q : g^t\}] \end{aligned} \tag{7}$$

Inequality (7) follows from theorem 2.7.1 regarding statistical and computational distance.

We compute

$$\Delta[\{x,y \xleftarrow{r} \mathbb{Z}_q : g^{xy}\}, \{t \xleftarrow{r} \mathbb{Z}_q : g^t\}] = \frac{1}{2}\sum_{z \in \langle g \rangle} \Big|\underset{x,y \xleftarrow{r} \mathbb{Z}_q}{\mathsf{Prob}}[g^{xy}=z] - \underset{t \xleftarrow{r} \mathbb{Z}_q}{\mathsf{Prob}}[g^t=z]\Big|. \tag{8}$$

We have that for some $z \in \mathbb{G}$

$$\begin{aligned}\underset{x,y \xleftarrow{r} \mathbb{Z}_q}{\mathsf{Prob}}[g^{xy}=z] &= \underset{x,y \xleftarrow{r} \mathbb{Z}_q}{\mathsf{Prob}}[x=0] \underset{x,y \xleftarrow{r} \mathbb{Z}_q}{\mathsf{Prob}}[g^{xy}=z|x=0] + \underset{x,y \xleftarrow{r} \mathbb{Z}_q}{\mathsf{Prob}}[x\neq 0] \underset{x,y \xleftarrow{r} \mathbb{Z}_q}{\mathsf{Prob}}[g^{xy}=z|x\neq 0] \\ &= \begin{cases} \frac{1}{q}\cdot 1 + \frac{q-1}{q}\cdot\frac{1}{q} = \frac{1}{q} + \frac{q-1}{q^2}, & z = g^0 \\ \frac{1}{q}\cdot 0 + \frac{q-1}{q}\cdot\frac{1}{q} = \frac{q-1}{q^2}, & z \neq g^0 \end{cases}\end{aligned} \tag{9}$$

By (8), (9),

$$\begin{aligned}\Delta[\{x,y \xleftarrow{r} \mathbb{Z}_q : g^{xy}\}, \{t \xleftarrow{r} \mathbb{Z}_q : g^t\}] &= \frac{1}{2}\Big(\Big|\underset{x,y \xleftarrow{r} \mathbb{Z}_q}{\mathsf{Prob}}[g^{xy}=g^0] - \underset{x,y \xleftarrow{r} \mathbb{Z}_q}{\mathsf{Prob}}[g^t=g^0]\Big| + \\ &\qquad \sum_{z \in G\setminus\{g^0\}} \Big|\underset{x,y \xleftarrow{r} \mathbb{Z}_q}{\mathsf{Prob}}[g^{xy}=z] - \underset{x,y \xleftarrow{r} \mathbb{Z}_q}{\mathsf{Prob}}[g^t=z]\Big|\Big) \\ &= \frac{1}{2}\Big(\Big|\Big(\frac{q-1}{q^2} + \frac{1}{q}\Big) - \frac{1}{q}\Big| + (q-1)\Big|\frac{q-1}{q^2} - \frac{1}{q}\Big|\Big) \\ &= \frac{q-1}{q^2}.\end{aligned} \tag{10}$$

By (7), (10) we get

$$|\delta - \delta'| \leq \frac{q-1}{q^2}, \tag{11}$$

which is $\mathsf{negl}(\lambda)$ since $q = \omega(poly(\lambda))$. This completes the proof of the claim. ■

6.5 Suitable Group Generators for the DDH Assumption

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

First consider $\mathbb{G}\langle g\rangle = \mathbb{Z}_p^*$ 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 6 that breaks the DDH assumption.

```
Algorithm A(G, q, g, a, b, c)
        if (a^{q/2} = 1 ∨ b^{q/2} = 1) ∧ (c^{q/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 $q = \mathsf{ord}(g)$ is even.

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

Let $\gamma = \langle \mathbb{G}, g, q, a, b, c\rangle$ 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

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

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 $c^{q/2} = (g^{xy})^{q/2} = 1$ as long as one of x or y is even. Thus,

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

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

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

Based on this information,

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

In an ideal situation, both probabilities are close to 1/2, so their difference is negligible. Since $\mathsf{Adv}^{\mathcal{A}} = 3/8$, $\mathcal{A}$ 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.

6.6 From Elements to Integers, a 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 $y \xleftarrow{r} \langle g\rangle$).

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 generally understand the structure of integers as opposed to elements of some arbitrary group.

One approach is to define a predicate V such that $\mathsf{Prob}_{x \leftarrow \langle g \rangle}[V(x) = 1] = 1/2$. 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.

We will instead, give a practical example for the popular order group $\mathbb{G} = QR(p)$, the order q subgroup of $\mathbb{Z}_p^*$ where $p = 2q + 1$, p, q are prime and $p \equiv 3 \mod 4$ and define the map

$$H \colon \mathbb{Z}_q \longrightarrow QR(p)$$

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

$$\begin{aligned} (x+1)^2 &\equiv (y+1)^2 \bmod p \\ (x+1)^2 - (y+1)^2 &\equiv 0 \bmod p \\ x^2 + 2x - 2y - y^2 &\equiv 0 \bmod p \\ (x-y)(x+y+2) &\equiv 0 \bmod p. \end{aligned}$$

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

$$\begin{aligned} x + y + 2 &\leq 2(q-1) + 2 = 2q \\ &< 2q + 1 \equiv 0 \bmod p. \end{aligned}$$

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

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

Using this, we can modify the key exchange protocol as is seen in Figure 7.

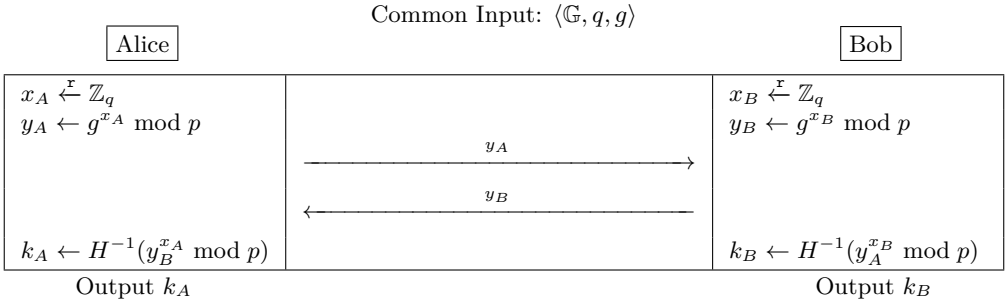


Figure 7: The modified Diffie-Hellman key exchange protocol where p is a large prime, g generates the group $QR(p)$ of order q, and $H \colon \mathbb{Z}_q \longrightarrow QR(p)$ by $x \mapsto (x+1)^2 \bmod p$.

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 q.

Exercise: We have shown how to derive a random element from $\mathbb{Z}_q$. This enables us to access cryptographic applications requiring a random integer modulo q 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 q.

It is interesting to note that in a λ-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.

6.7 Stronger Adversaries

While the Diffie-Hellman key exchange protocol, as given in Section 6.6, 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.

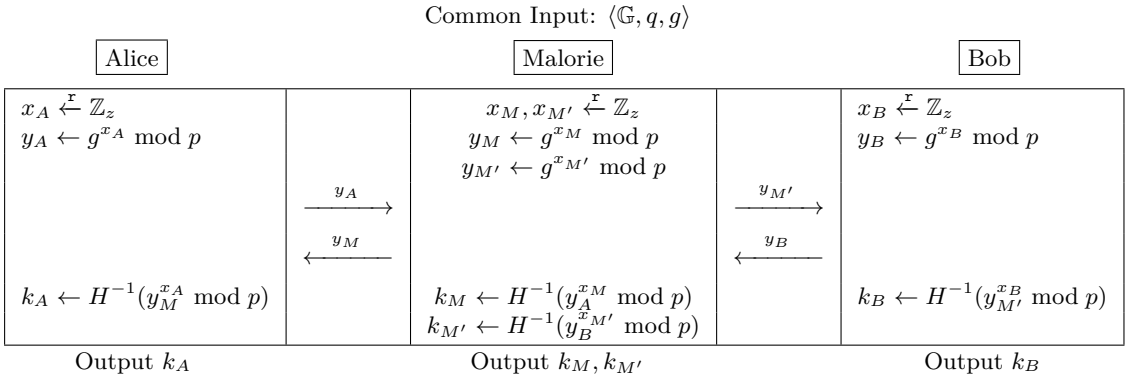


Figure 8: The "man-in-the-middle" attack on the Diffie-Hellman key exchange protocol.

7 Digital Signatures

A digital signature is a fundamental cryptographic primitive, technologically equivalent to a handwritten 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 7.0.1. A ***digital signature scheme*** is a triple of algorithms (Gen,Sign,Verify)[8] such that

- The key generation algorithm $\mathsf{Gen}\left(1^{\lambda}\right)$: Output the pair (vk, sk). We call vk the verification or public key and sk the signing of secret key.
- The signing algorithm $\mathsf{Sign}\left(vk, sk, M\right)$: Output σ. We call σ the digital signature of M, signed under the secret key sk.

[8] Gen and Sign are PPT algorithms, Verify is a deterministic polynomial-time algorithm.

- The verification algorithm $\mathsf{Verify}(vk, M, \sigma)$: Output `True` (or 1) if the signature is valid, `False` (or 0) otherwise.

A digital signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ must have the ***correctness*** and ***unforgeability*** properties.

- ***Correctness***: For any $M \in \{0,1\}^*$,

$$\underset{(vk,sk)\xleftarrow{\$}\mathsf{Gen}(1^\lambda)}{\mathsf{Prob}} [\mathsf{Verify}(vk, M, \mathsf{Sign}(vk, sk, M)) = 1] \geq 1 - \mathsf{negl}(\lambda) \quad .$$

- ***Unforgeability*** (informal): There exists no PPT adversary that can produce a valid message-signature pair without receiving it from external sources.

There are several versions of cryptographic games that model unforgeability, with different levels of security guarantees each. Here we introduce a version known as ***existential unforgeability against adaptively chosen message attacks*** ($\mathsf{EUF\text{-}CMA}$, Figure 9). In the adaptively chosen message attack, the adversary $\mathcal{A}$ wants to forge a signature for a particular public key (without access to the corresponding secret key) and has access to a signing oracle, which receives messages and returns valid signatures under the public key in question. Let Q be the set of messages for which $\mathcal{A}$ requested a signature from the signing oracle.

- $(vk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$
- $(M, \sigma) \leftarrow \mathcal{A}^{\mathsf{Sign}(vk,sk,\cdot)}$
- If $M \notin Q \wedge \mathsf{Verify}(vk, \sigma, M) = 1$, output 1
- Else, output 0

Figure 9: $\mathrm{Game}^{\mathcal{A}^{\mathsf{Sign}}}_{\mathsf{EUF-CMA}}(1^\lambda)$: Unforgeability against chosen message attacks.

We say that a digital signature scheme has the ***existential unforgeability*** property against ***adaptively chosen message attacks*** if

$$\forall \,\mathrm{PPT}\ \mathcal{A}, \mathsf{Prob}[\mathrm{Game}^{\mathcal{A}^{\mathsf{Sign}}}_{\mathsf{EUF-CMA}}(1^\lambda) = 1] \leq \mathsf{negl}(\lambda) \quad .$$

For brevity, we will simply call such schemes *secure*.

7.1 Trapdoor One-Way-Functions

A ***trapdoor one-way-function*** $f_e : X_e \mapsto Y_e$ with parameters $(e, z) \leftarrow \mathsf{Gen}(1^\lambda)$ is a function which satisfies the following:

- ***Easy to compute***: there exists a PPT algorithm that on input x returns $f_e(x)$.
- ***Hard to invert***: for every PPT algorithm $\mathcal{A}$,

$$\mathsf{Prob}[x \xleftarrow{\$} X_e;\ \mathcal{A}(e, f_e(x)) \in f_e^{-1}(f_e(x))] \leq \mathsf{negl}(\lambda).$$

- ***Easy to invert with trapdoor***: There exists PPT algorithm $\mathcal{T}$ such that

$$\mathcal{T}(e, z, f_e(x)) \in f_e^{-1}(f_e(x)).$$

Note that it must be $|X_e| \in \omega(\mathsf{poly}(\lambda))$, otherwise an adversary, on input $\langle e, y\rangle$, could simply try every $x \in X_e$ until $f_e(x) = y$.

7.2 Collision Resistant Hash Functions

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 to map to the same string (a.k.a. hash).

We need a family of hash functions in order to define what a collision-resistant hash function is. That is because the adversary can have hardwired a collision pair in his code and output it every time we ask him. Note that this is not a problem for the security of the one-way function, since in that case we ask the adversary for the inversion of a random element in the range of the function.

Thus, a family of hash functions $\mathcal{F} = \{\mathcal{H}_i : D_i \mapsto R_i\}_{i\in\mathcal{I}}$ is collision resistant if it satisfies the following:

- ***Easy to sample***: There exists a PPT algorithm Gen, such that for all $\lambda \in \mathbb{N}, \mathsf{Gen}(1^\lambda) \in \mathcal{I}$.
- ***Easy to compute***: There exists a PPT algorithm that on input $i \in \mathcal{I}, x \in D_i$ returns $\mathcal{H}_i(x)$.
- ***Compressing***: For all $i \in \mathbb{N}, |R_i| < |D_i|$.
- ***Collision resistant***: For every PPT algorithm $\mathcal{A}$, for all $\lambda \in \mathbb{N}$:

$$\mathsf{Prob}[i \leftarrow \mathsf{Gen}(1^\lambda); (x, y) \leftarrow \mathcal{A}(1^\lambda, \mathcal{H}_i) : (x \neq y) \wedge \mathcal{H}_i(x) = \mathcal{H}_i(y)] \leq \mathsf{negl}(\lambda)$$

7.3 Random Oracles

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 hash function as a random oracle, it is said to be secure in the ***Random Oracle Model***.

Figure 10 illustrates how a hash function H is modeled as a random oracle.

- Given $M \notin$ *History*, choose $t \xleftarrow{r} Y_e$ and add (M, t) to *History*. Return t.
- Given M such that $(M, t) \in$ *History* for some t, return t.

Figure 10: A random oracle, where *History* represents the set of all (*input*,*output*) pairs previously served by the oracle.

7.4 Digital Signatures from Trapdoor One-Way-Functions

Definition 7.4.1. Let $H : \{0,1\}^* \mapsto Y_e$ be a collision resistant hash function and $f_e : X_e \mapsto Y_e$ a trapdoor one way function with parameter generation algorithm $\mathsf{Gen}_{\mathsf{TOWF}}$ and trapdoor algorithm T. We define the following digital signature scheme:

- $\mathsf{Gen}(1^\lambda)$: $(e, z) \leftarrow \mathsf{Gen}_{\mathsf{OWF}}(1^\lambda)$. Let $(vk, sk) = (e, z)$. Output (vk, sk).
- $\mathsf{Sign}(vk, sk, M)$: $h \leftarrow H(M); \sigma \leftarrow T(vk, sk, h)$. Output σ.
- $\mathsf{Verify}(vk, M, \sigma)$: If $f_e(\sigma) = H(M)$ output `True`; otherwise output `False`.

It is straightforward to see that if f_e is a trapdoor one-way-function, then the scheme of 7.4.1 satisfies correctness:

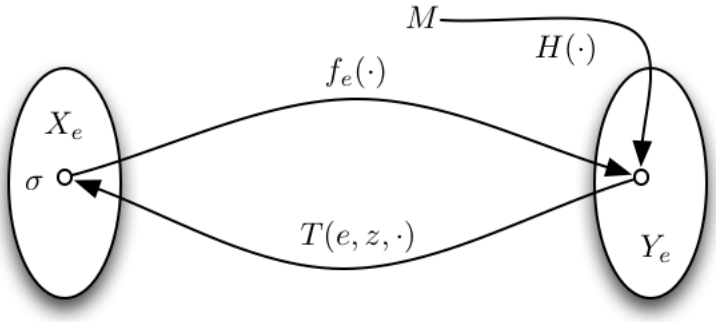


Figure 11: It easy to verify that $f_e(\sigma) = H(M)$, but is difficult to find a pre-image of M.

Proof of Correctness for 7.4.1. For any (vk, sk) returned by $\mathsf{Gen}\left(1^\lambda\right)$ and for every $M \in \{0,1\}^*$, it is:

$$\mathsf{Verify}\left(vk, M, \mathsf{Sign}\left(vk, sk, M\right)\right) = \mathsf{Verify}\left(vk, M, T\left(vk, sk, H\left(M\right)\right)\right)$$

Since $f_e\left(T\left(pk, sk, H\left(M\right)\right)\right) = H\left(M\right)$, Verify will return `True`. ■

We will now prove that the digital signature scheme is secure in the Random Oracle Model (i.e. H will be modeled as a Random Oracle) under the assumption that f_e is trapdoor one-way-function.

Theorem 7.4.1. *Consider the Digital Signature scheme defined in 7.4.1 with security parameter 1^λ for some $\lambda \in \mathbb{N}$. Suppose that $f_e : X_e \mapsto Y_e$ is a bijective (a.k.a. one-to-one) trapdoor one-way-function with $|X_e| = |Y_e| \geq 2^\lambda$ and that $H : \{0,1\}^* \mapsto Y_e$ is a Random Oracle. Then for every PPT algorithm $\mathcal{A}$ that breaks the security of the scheme, i.e.*

$$\forall\, \mathsf{PPT}\ \mathcal{A} : \exists \alpha > \mathsf{negl}(1^\lambda) \text{ with } \mathsf{Prob}\left[\mathrm{Game}^{\mathcal{A}^{Sign}}_{\mathsf{EUF-CMA}}\left(1^\lambda\right) = 1\right] = \alpha \ ,$$

there exists PPT algorithm $\mathcal{B}$ that violates the one-way property of f_e, i.e.

$$\exists\, \mathsf{PPT}\ \mathcal{B} : \mathsf{Prob}\left[x \xleftarrow{\$} X_e; B\left(e, f_e\left(x\right)\right) = x\right] \geq \frac{1}{q_H} \cdot \left(\alpha - \frac{1}{2^\lambda}\right) \ ,$$

where q_H is the number of queries $\mathcal{A}$ makes to the random oracle H.

Proof. Let $\mathcal{A}$ be a PPT algorithm that finds a forged signature with probability a. We will specify a PPT algorithm $\mathcal{B}$ that violates the one-way property of f_e.

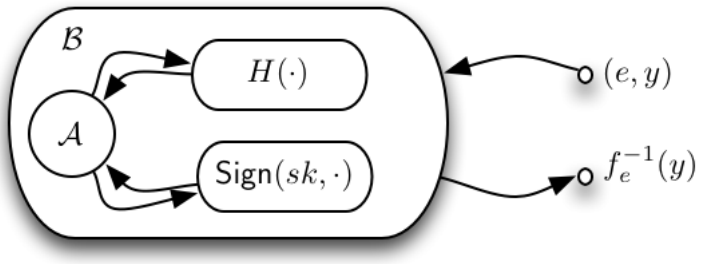


Figure 12: The attacker $\mathcal{B}$ must simulate H and Sign to use attacker $\mathcal{A}$.

Let $(e, z) \leftarrow \mathsf{Gen}_{\mathsf{TOWF}}\left(1^\lambda\right), x \xleftarrow{\$} X_e$ and $y = f_e\left(x\right)$. Note that, since f_e is a bijection, $\nexists x' \neq x : f_e\left(x'\right) = y$. $\mathcal{B}$ is given input $\langle e, y\rangle$ and its goal is to find x. To do this, $\mathcal{B}$ gives e to $\mathcal{A}$ as verification key. $\mathcal{A}$ 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 12 illustrates how $\mathcal{B}$ operates when given access to $\mathcal{A}$.

First suppose $\mathcal{A}$ does not invoke the signing algorithm Sign, so $\mathcal{A}$ produces (M, σ) after making q_H queries to the random oracle H. $\mathcal{B}$ answers these queries by simulating H as in Figure 13. We will now justify why $\mathcal{B}$ departs from the Random Oracle description of Figure 10. We will prove that this departure does not invalidate the Random Oracle Model as the last step of this proof. By our assumption, σ is a valid forged signature for M with probability a. If it is valid (event D), it is $\sigma = f_e^{-1}(H(M))$. If furthermore $H(M) = y$ (event L), then σ is a pre-image of y.

Observe that, if $\mathcal{B}$ simulates the Random Oracle as in 10, and given that D takes place, the event L happens with negligible probability, as

$$\mathsf{Prob}[L|D] = \mathsf{Prob}[H(M) = y|\, D] = \frac{1}{|Y_e|} \leq \frac{1}{|X_e|} \leq \mathsf{negl}(1^\lambda) \ .$$

In order to obtain this result with non-negligible probability, $\mathcal{B}$ deviates from Figure 10 when answering the random oracle queries of $\mathcal{A}$. $\mathcal{B}$ now chooses $j \xleftarrow{\$} \{1, \ldots, q_H\}$ and answers the jth query with y. More specifically, the modified random oracle is parametrized by q_H and y and operates as follows:

Choose $j \xleftarrow{\$} \{1, 2, ..., q_H\}$.
- Given M such that $\nexists(M, \_) \in \mathit{History}$: If this is the jth distinct query, set $t = y$, else choose $t \xleftarrow{\$} Y_e$. Add (M, t) to *History*. Return t.
- Given M such that $(M, t) \in \mathit{History}$ for some t, return t.

Figure 13: A modified random oracle simulation as used by algorithm $\mathcal{B}$ to "plug-in" a challenge y into the oracle's responses.

Consider the event E, that $(M, \_) \in \mathit{History}$ when the internal execution of $\mathcal{A}$ by $\mathcal{B}$ is over, and the event $\neg E$, the complement. Also let S be the event that $\mathcal{A}$ successfully produces a forged signature. The theorem hypothesis states that $\mathsf{Prob}[S] = a$. On the other hand, observe that $\mathsf{Prob}[S|\neg E] \leq \frac{1}{|Y_e|} \leq 2^{-\lambda}$. This is the case since given the event $\neg E$, the adversary has not asked M to H and thus the value of $H(M)$ is undetermined until the final step of $\mathcal{B}$ takes place. Since the adversary has already produced σ, $\mathsf{Prob}[H(M) = f_e(\sigma)\,|\neg E] = \frac{1}{|Y_e|} \leq 2^{-\lambda}$.

Given now the event E, there is a $\frac{1}{q_H}$ chance the random oracle simulation 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. If additionally the event S took place, the algorithm $\mathcal{B}$ would have successfully recovered a pre-image of y. We call this the event V, i.e. $V = S \wedge G$. We next provide a lower bound on V. First we have that $\mathsf{Prob}[S|\neg E] \leq 2^{-\lambda}$, therefore

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

Based on this we obtain the lower bound

$$\mathsf{Prob}[S \wedge E] = \mathsf{Prob}[S] - \mathsf{Prob}[S \wedge \neg E] \geq a - 2^{-\lambda} \ .$$

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

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

Due to the independence of the events S, G in the conditional space E (verify!) and because $\mathsf{Prob}[G|E] = \frac{1}{q_H}$, we have that

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

This completes the argument in case $\mathcal{A}$ makes no queries to the signing oracle. We next consider the general case where the number of queries to the signing oracle, q_S, is a polynomial in λ. We denote the queries to the signing oracle by $M_1, \ldots, M_{q_S}$. $\mathcal{B}$ must answer such queries in a way that is consistent with the random oracle queries: if $\mathcal{B}$ returns σ_i, it holds that $\sigma_i = f_e^{-1}(H(M_i))$, so $f_e(\sigma_i) = H(M_i)$. This implies that $(M_i, f_e(\sigma_i))$ is in *History*. We can accomodate this by again modifying the simulation of H performed by $\mathcal{B}$ as seen in Figure 14.

Choose $j \xleftarrow{\$} \{1, 2, ..., q_H\}$.
- Given M such that $\nexists(M, \_, \_) \in$ *History*: If this is the jth distinct query, set $t = y$, $\rho = \diamond$, else choose $\rho \xleftarrow{r} X_e$ and set $t = f_e(\rho)$.
  Enter (M, t, ρ) in *History*. Return t.
- Given M such that $(M, t, \rho) \in$ *History* for some t, return t.

Figure 14: 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 f_e map.

Now when asked to sign M_i, $\mathcal{B}$ first asks the random oracle for M_i (this does not count towards the j RO queries) and then consults the *History* table for a record of the form (M_i, t_i, ρ_i). Unless $\rho_i = \diamond$, it proceeds to answer the signing query with ρ_i. Observe that the signing oracle simulation is perfect as long as $\rho_i \neq \diamond$, since $f_e(\rho_i) = t_i$, i.e., ρ_i is the pre-image of $t_i = H(M_i)$. The case $\rho_i = \diamond$ means that the guess of $\mathcal{B}$ for j 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 of $\mathcal{B}$ will fail. However, for the event $\rho_i = \diamond$, call it F, it holds that $G \cap F = \emptyset$.

We can again deduce that $\mathsf{Prob}[S \wedge \neg E] \leq 2^{-\lambda}$ using similar arguments to the ones in the case of no queries to the signing oracle, therefore once more it is $\mathsf{Prob}[S \wedge E] \geq a - 2^{-\lambda}$. We also observe that, since if event F happens the signature forgery always fails, it is $\mathsf{Prob}[S \wedge G \wedge E \wedge F] = 0 \Rightarrow \mathsf{Prob}[S \wedge G \wedge E \wedge \neg F] = \mathsf{Prob}[S \wedge G \wedge E]$. Also the events G and S are independent. Thus we have

$$\mathsf{Prob}[V] \geq \mathsf{Prob}[S \wedge G \wedge E \wedge \neg F] = \mathsf{Prob}[S \wedge G \wedge E] =$$
$$\mathsf{Prob}[S \wedge G | E] \cdot \mathsf{Prob}[E] = \mathsf{Prob}[S|E] \cdot \mathsf{Prob}[G|E] \cdot \mathsf{Prob}[E] =$$
$$\mathsf{Prob}[S \wedge E] \cdot \mathsf{Prob}[G|E] \geq \left(a - 2^{-\lambda}\right) \frac{1}{q_H}$$

To finish the proof we need to ensure that the adversary $\mathcal{A}$ cannot distinguish the random oracle as defined in Figure 14 from the one defined in Figure 10. Indeed, observe that now $q_H - 1$ of the values returned to $\mathcal{A}$ are of the form $f_e(\rho)$, as opposed to random values t selected uniformly from Y_e. This nevertheless turns out not to be an issue since f_e is a bijection and as a result $f_e(\rho)$ is a random variable that is uniformly distributed over Y_e given that ρ is uniformly distributed over X_e. As for the jth query, we recall that the input y to $\mathcal{B}$ is distributed uniformly at random over Y_e (since $y = f_e(x)$ and $x \xleftarrow{\$} X_e$). It is thus chosen from the exact same distribution as the jth query to the original Random Oracle (Figure 10). ■

7.5 The RSA Function: The eth 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 ***eth power map*** of x. She then sends $y = E(x)$ to Bob, who in order to see the message, must compute the eth root of y. This is believed to be hard, as we discussed in Section 6.2. If Bob selects n and e appropriately however, there is an alternate method. We will see that Bob can apply the dth 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 7.5.1. For $n \in \mathbb{N}$, the ***Euler function*** $\varphi(n)$ counts the number of integers in $\mathbb{Z}_n$ relatively prime to n:

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

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

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

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 φ to a composite integer $n = p_1^{e_1} \cdots p_j^{e_j}$ falls from the fact that φ 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 $\mathbb{Z}_n^* = \{a \in \mathbb{Z}_n \colon \gcd(a, n) = 1\}$ 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 eth power map

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

defined by $x \mapsto x^e \bmod n$ is invertible. In particular, when $ed \equiv 1 \bmod \varphi(n)$, the dth power map D inverts E. If Bob choses 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 eth 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.

Proposition 7.5.1. *The function $E(x) : \mathbb{Z}_n^* \mapsto \mathbb{Z}_n^*$ is trapdoor one-way function.*

Proof.

- It is easy to compute.
- It is hard to invert if we do not know the factorization of n (RSA assumption).
- There exist a PPT algorithm, that computes $D(y)$.

■

7.6 RSA Digital Signatures

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

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

where $\{0,1\}^* = \bigcup_{k=0}^{\infty} \{0,1\}^k$. Note that in practice, the range of H will be $\mathbb{Z}_n \setminus \{0\}$ and we will rely on the fact that $1 - \dfrac{\mathbb{Z}_n^*}{\mathbb{Z}_n \setminus \{0\}}$ is negligible to argue that effectively sampling from $\mathbb{Z}_n \setminus \{0\}$ is indinstinguishable sampling from $\mathbb{Z}_n^*$.

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 $d \equiv e^{-1} \bmod \varphi(n)$. 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, σ), verify that $\sigma^e = H(M) \bmod n$. If equality holds, the result is `True`; otherwise the result is `False`.

8 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 π.
- The verifier $\mathcal{V}$ has R, x, and π.

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 π.

8.1 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 openent 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

15. 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.

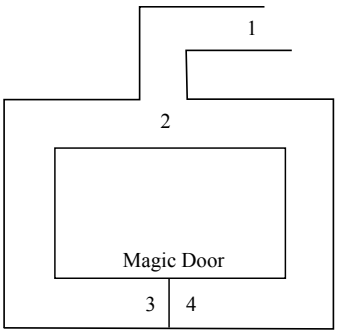


Figure 15: The door between Points 3 and 4 can only be opened using a secret password.

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

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

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

Another language is $\mathsf{SAT} = \{\langle \Phi\rangle : \Phi \text{ is a satisfiable boolean formula}\}$. One can check in polynomial-time that a set of variables assigned to $\Phi \in SAT$ satisfies Φ. Zero-knowledge proofs can be used to prove a specific element is in CLIQUE or SAT. We will address how in Section 8.5

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 8.3.

8.2 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 8.2.1. Let $\langle \mathcal{P}, \mathcal{V}\rangle$ be a pair of interactive programs running polynomial-time in public input x. Define $\mathsf{out}^{\mathcal{P}}_{\mathcal{P},\mathcal{V}}(x, w, z)$ 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); $\mathsf{out}^{\mathcal{V}}_{\mathcal{P},\mathcal{V}}$ is similarly defined for $\mathcal{V}$. The PPT interactive protocol $\langle \mathcal{P}, \mathcal{V}\rangle$ is a ***zero-knowledge proof*** for a language $\mathsf{L} \in NP$ if the following properties hold.

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

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

  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

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

  Then it holds that $\pi_{x,w,z}$ is non-negligible in $|x|$ implies that $\widetilde{\pi}_{x,w,z}$ is non-negligible in $|x|$.
- **(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 $\mathsf{out}^{\mathcal{V}^*}_{\mathcal{P},\mathcal{V}^*}(x, w, z)$ are statistically indistinguishable for all strings z:

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

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 must 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).

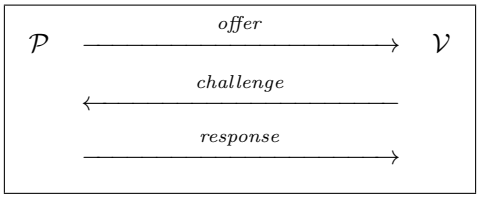
We note that our formulation of soundness is a bit more restrictive (albeit much simpler) than previous formulations in the literature, as it will fail protocols that allow a substantial cheating probability for the prover (e.g., 1/2). In most interesting cases such protocols can be made to satisfy our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

8.3 The Schnorr Protocol

One classic three-move protocol that exhibits the properties of a zero-knowledge proof is the Schnorr protocol, also know as the Σ-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 $h = g^w \bmod p$ 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 $\mathsf{DLOG} = \{\langle \langle p, m, g \rangle, h \rangle : h = g^w \bmod p \text{ for some } w \in \mathbb{Z}_m\}$. ($\mathsf{DLOG}$ stands for "discrete logarithm".) Under the Schnorr protocol, there is a very efficient way to prove any statement $\langle \langle p, m, g \rangle, h \rangle$ is in DLOG without revealing $w = \log_g h$.

1. $\mathcal{P}$ choses $t \xleftarrow{r} \mathbb{Z}_m$ and sends $y = g^t$ to $\mathcal{V}$.
2. $\mathcal{V}$ chooses a challenge $c \xleftarrow{r} \mathbb{Z}_m$ and sends c to $\mathcal{P}$.
3. $\mathcal{P}$ computes $s = t + wc \bmod m$ 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. $\mathcal{P}(\mathsf{first}, \langle p, m, g\rangle, h)$ outputs $\langle y, \mathsf{aux}\rangle$
2. $\mathcal{P}(\mathsf{second}, \langle p, m, g\rangle, c, \mathsf{aux})$ outputs $\langle s\rangle$

where 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 $\rho_1 \xleftarrow{r} \{0,1\}^{\lambda_1}$ be the coin tosses required by the first step of $\mathcal{P}$. Fix the randomness of $\mathcal{P}$ with ρ_1 and simulate $\mathcal{P}(\mathsf{first}, \langle p, m, g\rangle, h)$ to obtain y.
2. Choose $c \xleftarrow{r} \mathbb{Z}_m$.
3. Let $\rho_2 \xleftarrow{r} \{0,1\}^{\lambda_2}$ be the coin tosses required by the second step of $\mathcal{P}$. Simulate $\mathcal{P}(\mathsf{second}, \langle p, m, g\rangle, c, \mathsf{aux})$ with fixed randomness ρ_2 to obtain s.
4. Choose $c' \xleftarrow{r} \mathbb{Z}_m$ and $\rho_2' \xleftarrow{r} \{0,1\}^{\lambda_2}$. 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 8.3.1 (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*

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

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

$$|A| = |A'| + |A \setminus A'| < \frac{\alpha}{2}\,|X \times Y| + |A \setminus A'|\,. \tag{12}$$

For any $(x,y) \in A \setminus A'$, we have that $k_x \leq (\alpha/2)\,|Y|$. Since there are only $|X|$ distinct xs, $|A \setminus A'| \leq (\alpha/2)\,|X|\,|Y|$. From (12) 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

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

If the prover is successful with at least probability α, 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 8.3.1, 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

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

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 $t, c \xleftarrow{\mathrm{r}} \mathbb{Z}_m$. 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 $(c', \sigma) \leftarrow \mathsf{Commit}(c)$. $\mathcal{V}$ then sends c' to the prover. When $\mathcal{P}$ sends y, the verifier returns (c, σ) to open the commitment. If $\mathsf{Verify}(c, c', \sigma) = 0$, 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 $(y', \sigma) \leftarrow \mathsf{Commit}(y)$ and sends y' to $\mathcal{V}$. Once $\mathcal{V}$ returns c, $\mathcal{P}$ sends (y, σ, s) to open the commitment. If $\mathsf{Verify}(y, y', \sigma) = 0$, 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.

8.4 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 $H\colon \{0,1\}^* \longrightarrow \mathbb{Z}_m$ 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 16 illustrates how $\mathcal{V}^*$ interacts with H.

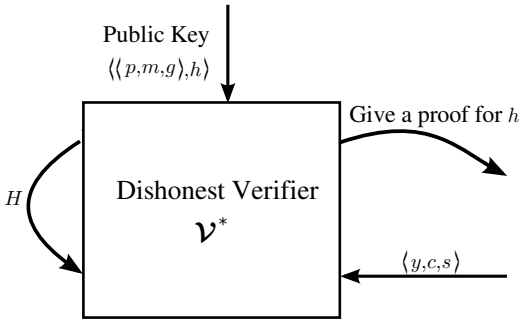


Figure 16: The simulation of a dishonest verifier $\mathcal{V}^*$ in the Random Oracle Model.

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 $(y, c') \in History$ 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 $\mathsf{Prob}[A] = \alpha$ denote the probability that the resulting conversation is accepting. Let $\mathsf{Prob}[Q_i] = \beta_i$ represent the probability that the dishonest prover completes the ith conversation with $1 \leq i \leq q_H$. Define $\mathsf{Prob}[A \cap Q_i] = \alpha_i$, then

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

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

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

so $\mathsf{Prob}[A \mid Q_i] = \alpha_i/\beta_i$. From our calculations in Section 8.3, we obtain

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

The overall probability is calculated as follows.

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

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

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

8.5 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*** π 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 ith vertex is connected to the jth vertex, the ij entry in the matrix is 1; otherwise it is 0. Given a permutation π over $\{1, ..., n\}$ and a graph G defined by its adjacency matrix (a_{ij}), we define the permuted graph G^π 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 π of the vertices with the special property that the graph G^π includes the edges $(1,2), (2,3), ..., (n-1,n), (n,1)$.

If π is a Hamiltonian cycle for a graph G and π' 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 17 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.

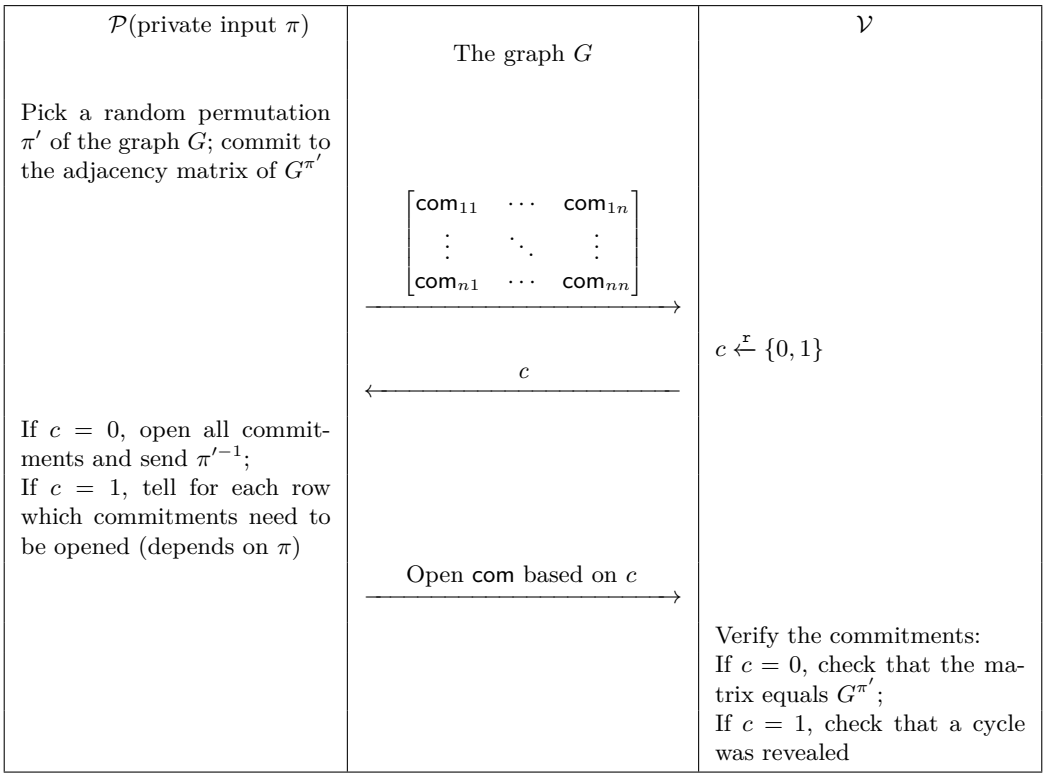


Figure 17: A proof of knowledge for a Hamiltonian cycle. In addition to committing to the adjacency matrix of $G^{\pi'}$, the prover must make a separate commitment com_{ij} to each entry in the matrix.

8.6 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 18.

Theorem 8.6.1. *The conjunction of two honest-verifier zero-knowledge proofs satisfies the soundness and HVZK properties.*

8.7 The Disjunction of Two Zero-Knowledge Proofs

In Section 8.1 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 19 depicts the execution of a disjunction.

Theorem 8.7.1. *The disjunction of two honest-verifier zero-knowledge proofs satisfies the soundness and HVZK properties.*

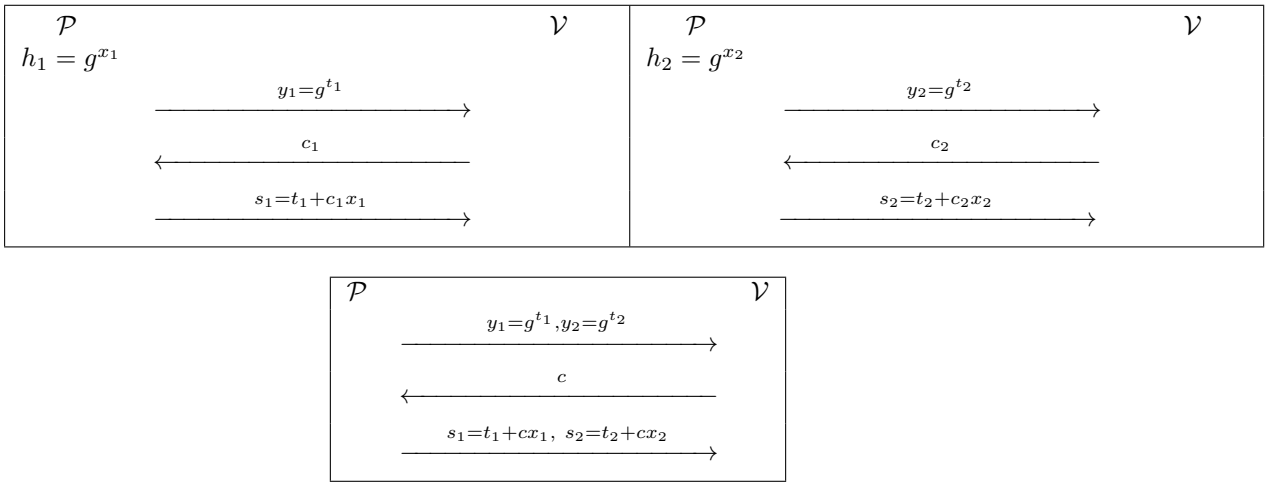


Figure 18: The conjunction of two zero-knowledge proofs for the discrete logarithm.

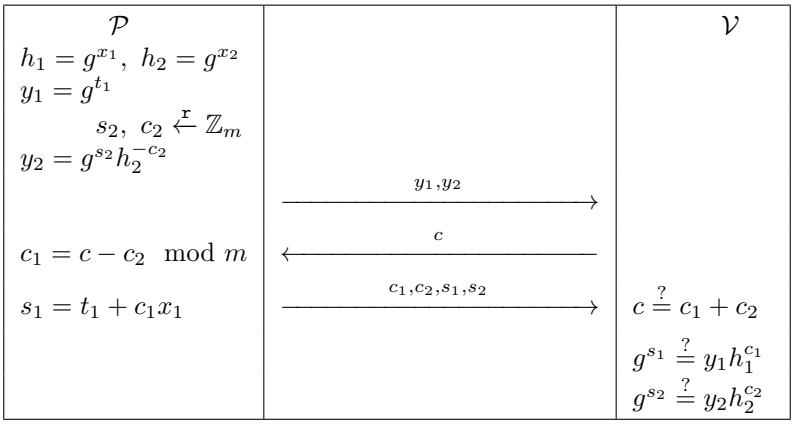


Figure 19: The disjunction of two zero-knowledge proofs for the discrete logarithm showing how the prover can show the verifier he knows one of the two discrete logarithms of h_1, h_2 (without revealing which one). In this case the prover $\mathcal{P}$ knows a witness for h_1.

9 Public-Key Encryption

In Section 4 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 9.0.1. 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 $\mathcal{E} \colon \mathcal{K}_p \times \mathcal{M} \longrightarrow \mathcal{C}$ that preserves λ: $\mathcal{E}(\mathcal{P}_\lambda \times \mathcal{M}_\lambda) \subseteq \mathcal{C}_\lambda$
- An efficient decryption algorithm $\mathcal{D} \colon \mathcal{K}_s \times \mathcal{C} \longrightarrow \mathcal{M}$ that preserves λ: $\mathcal{D}(\mathcal{S}_\lambda \times \mathcal{C}_\lambda) \subseteq \mathcal{M}_\lambda$
- An efficient key generation algorithm $\mathcal{G} \colon \mathbb{N} \longrightarrow \mathcal{P} \times \mathcal{S}$ that preserves λ: $\mathcal{G}(1^\lambda)$.

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

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

9.1 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 chose-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 9.1.1. A public-key cyrptosystem is AON-CPA secure provided

$$\mathsf{Prob}[\mathcal{A}(pk, c) = M : \langle pk, sk\rangle \leftarrow \mathcal{G}(1^\lambda),\ c \leftarrow \mathcal{E}(pk, M),\ M \xleftarrow{\mathrm{r}} \mathcal{M}_\lambda]$$

is negligible in λ 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.

9.2 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 indistinguishablility. We can model this by the following game, $\mathrm{Game}^{\mathcal{A}}_{\mathsf{IND\text{-}CPA}}(1^\lambda)$.

1. $\langle pk, sk\rangle \leftarrow \mathcal{G}(1^\lambda)$
2. $\langle \mathsf{aux}, M_0, M_1\rangle \leftarrow \mathcal{A}(\mathsf{play}, pk)$ for $M_0 \neq M_1$
3. $b \xleftarrow{\mathrm{r}} \{0, 1\}$
4. $c \leftarrow \mathcal{E}(pk, M_b)$
5. $b^* \leftarrow \mathcal{A}(\mathsf{guess}, \mathsf{aux}, c)$
6. If $b = b^*$ output 1; otherwise 0.

Figure 20 illustrates this game.

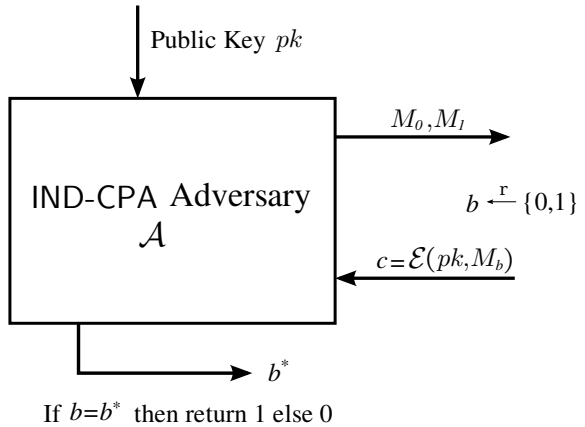


Figure 20: The IND-CPA attack. If $b = b^*$, we say the adversary wins the game.

Definition 9.2.1. A public-key cryptosystem is IND-CPA secure if for all PPT adversaries $\mathcal{A}$,

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

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

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

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

$\mathcal{D}(sk, c)$: $M = c^d \bmod n$
output M

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.

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

$\mathcal{E}(pk, M)$: $M \in \{0,1\}^{\lambda-\lambda_0}$
$r \in \{0,1\}^{\lambda_0}$
$M' = \mathsf{bit2integer}(r\|M)$
compute $c = M'^e \bmod n$
output c

$\mathcal{D}(sk, c)$: $M' = c^d \bmod n$
$(r\|M) = \mathsf{integer2bit}(M')$
output M

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.

9.3 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 $\mathbb{G} = \langle g\rangle$ of prime order q.

$\mathcal{G}(1^\lambda)$: $\langle pk, sk\rangle \leftarrow \mathcal{G}(1^\lambda)$
$x \xleftarrow{r} \mathbb{Z}_q, h = g^x$
$pk = \langle\langle \mathbb{G}, q, g\rangle, h\rangle$
$sk = x$

$\mathcal{E}(pk, M)$: $M \in \langle g\rangle$
$r \xleftarrow{r} \mathbb{Z}_q$
compute $U = g^r$, $V = h^r M$
output $\langle G, H\rangle$

$\mathcal{D}(sk, U, V)$: compute $M = V/U^x$
output M

Correctness Proving correctness is simple: We start with $\mathcal{D}(sk, \mathcal{E}(pk, M))$ which we expand to $\mathcal{D}(x, U, V)$ which will return V/U^x. But $V/U^x = (h^r M)/g^{rx} = (h^r M)/h^r = M$.

Security 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

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

for a nonnegligible α, we will construct a PPT algorithm $\mathcal{A}$ that can distinguish whether a tuple $\langle h, G, H\rangle$ is a DDH tuple or a random tuple.

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

1. $\langle \mathsf{aux}, M_0, M_1\rangle \leftarrow \mathcal{B}(\mathsf{play}, pk, h)$ where $pk = \langle\langle \mathbb{G}, q, g\rangle, h\rangle$
2. $b \xleftarrow{r} \{0, 1\}$
3. $c \leftarrow \langle G, H \cdot M_b\rangle$
4. $b^* \leftarrow \mathcal{B}(\mathsf{guess}, \mathsf{aux}, c)$
5. If $b = b^*$ output 1; otherwise 0.

Figure 21 illustrates how $\mathcal{A}$ uses the attacker $\mathcal{B}$ to break the DDH assumption.

We will first examine how $\mathcal{A}$ works in the case where it receives a DDH tuple. It is clear to see that in that case, $G, H \cdot M_b$ is indeed a valid ElGamal ciphertext with regard to public key h and message M_b. We thus expect $\mathcal{B}$ to produce the correct value of b with good probability.

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

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

Now, we examine the case where $\mathcal{A}$ receives a random tuple instead. If v contains a random tuple, $v = \langle\langle \mathbb{G}, q, g\rangle, g^x, g^y, g^z\rangle$. This will produce a ciphertext with somewhat unpredictable structure (and in most cases, one that does not decrypt to M_0 or M_b). As such, the behaviour of $\mathcal{B}$ can be unpredictable as well. The key point that we will establish is that $\mathcal{B}$ behaves the same regardless of b. We do not need to make any other claims about its behaviour, as that is enough to prove that b and b^* match with probability exactly 1/2.

For any $w \in \langle g\rangle$, we can find a unique z_0 such that $w = g^{z_0} M_b$ or $z_0 = \log_g(w/M_b)$ as well as a unique z_1 such that $w = g^{z_1} M_b$ i.e. $z_1 = \log_g(w/M_b)$. 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.

$$\mathop{\mathsf{Prob}}_{v \leftarrow \langle\langle \mathbb{G}, q, g\rangle, g^x, g^y, g^z\rangle}[\mathcal{A}(v) = 1] = \mathsf{Prob}[b = b^*] = \frac{1}{2}.$$

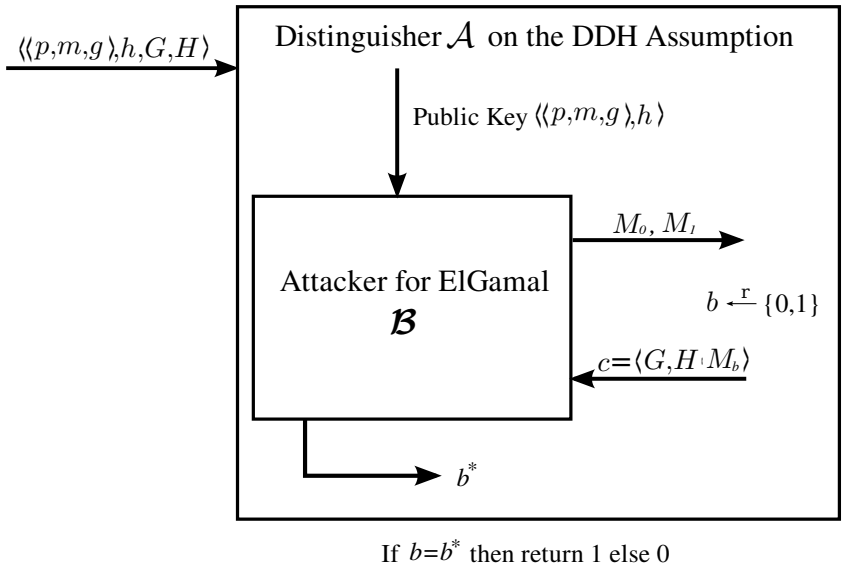


Figure 21: If an attacker $\mathcal{B}$ decodes a message in ElGamal with a nonnegligible probability, we can construct a PPT distinguisher $\mathcal{A}$ that breaks the DDH assumption. If $b = b^*$, $\mathcal{A}$ is successful.

Based on this,

$$\left| \underset{v \leftarrow \langle\langle \mathbb{G},q,g\rangle, g^x, g^y, g^{xy}\rangle}{\mathsf{Prob}} [\mathcal{A}(v) = 1] \quad - \underset{v \leftarrow \langle\langle \mathbb{G},q,g\rangle, g^x, g^y, g^z\rangle}{\mathsf{Prob}} [\mathcal{A}(v) = 1] \right| \geq \alpha.$$

Because α is nonnegligible, this violates the DDH assumption as desired.

10 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.

10.1 Game Basics

Lemma 10.1.1. *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 10.1.1. Let G be a deterministic program that takes λ-bit inputs and terminates after v steps. At each step, G employs a random variable ρ_i sampled uniformly from the domain R_i. Each space R_i is parameterized by the λ-bit input and may depend on previous variables: $R_i = \{0,1\}^{s_i(\lambda)}$ for $s_i \colon \mathbb{N} \longrightarrow \mathbb{N}$ and $1 \leq i \leq v$. The ith step of G has the form

$$y \leftarrow f(y_1, \ldots, y_d, \rho_1, \ldots, \rho_i); \quad \rho_i \xleftarrow{r} R_i^{y_1,\ldots,y_d}$$

where y is a variable, $y_1, \ldots, y_d$ 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 $\langle \rho_1, \ldots, \rho_v \rangle$).

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

	Game G_0 on Input 1^λ	Random Variables
1.	$p \leftarrow \mathsf{PRIME}(1^\lambda, \rho)$	$\rho \xleftarrow{\mathtt{r}} \mathsf{Coins}$
2.	$x \leftarrow \sum_{i=0}^{\lambda-1} 2^i \rho_{i+1}$	$\rho_1, \ldots, \rho_\lambda \xleftarrow{\mathtt{r}} \{0,1\}$
3.	output the least significant bit of $x \bmod p$	

10.2 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 10.2.1 similar to [2]. That is, we refrain from using any special syntactical conditions and take into account the special syntax properties of the underlying game program.

Lemma 10.2.1. (First Game-Playing Lemma). *Consider two games, G and G' defined over the same random variables $\rho_1, \ldots, \rho_v$ in $R_1, \ldots, R_v$. If $\mathsf{Prob}[T \cap \neg F] = \mathsf{Prob}[T' \cap \neg F]$ for some event F, then $|\mathsf{Prob}[T] - \mathsf{Prob}[T']| \leq \mathsf{Prob}[F]$.*

Proof. It holds that

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

So by our assumption,

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

It follows then that

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

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

Example. Suppose we add another step to game G_0 to obtain game G_0'.

	Game G_0' on Input 1^λ	Random Variables
1.	$p \leftarrow \mathsf{PRIME}(1^\lambda, \rho)$	$\rho \xleftarrow{\mathtt{r}} \mathsf{Coins}$
2.	$x \leftarrow \sum_{i=0}^{\lambda-1} 2^i \rho_{i+1}$	$\rho_1, \ldots, \rho_\lambda \xleftarrow{\mathtt{r}} \{0,1\}$
3.	if $x < 2^{\lfloor \lambda/2 \rfloor}$ then $x \leftarrow 2x - p$	
4.	output the least significant bit of $x \bmod p$	

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 $|\mathsf{Prob}[T_0] - \mathsf{Prob}[T_0']| \leq \mathsf{Prob}[F]$. It is then easy to verify that $\mathsf{Prob}[F] \leq 2^{\lfloor \lambda/2 \rfloor}$.

10.3 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 10.3.1. Suppose G is a game and the variables $y_1, \ldots, y_d$ are defined in G's syntactic description up to the ith step. Let $\widetilde{y}_1, \ldots, \widetilde{y}_d$ be values in the respective domains of $y_1, \ldots, y_d$. Let K be the event $\wedge_{i=1}^{d}(y_i = \widetilde{y}_i)$ 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 $y_1, \ldots, y_d$ are substituted by $\widetilde{y}_1, \ldots, \widetilde{y}_d$.
3. An initial dummy step is added to $G[K]$ that draws the random variables $\rho_1, \ldots, \rho_i$ from their respective spaces following their respective distributions conditioned on the event K.

Example. Consider the following game where 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^λ	Random Variables
1.	$p \leftarrow \mathsf{PRIME}(1^\lambda, \rho)$	$\rho \xleftarrow{r} \mathsf{Coins}$
2.	$x \leftarrow b + at \bmod p$	$a \xleftarrow{r} [p],\ t \xleftarrow{r} [p-1], b \xleftarrow{r} \{0,1\}$
3.	$c \leftarrow f_1(p, x; \rho_1)$	$\rho_1 \xleftarrow{r} \mathsf{Coins}_1$
4.	$y \leftarrow c + t \bmod p$	
5.	$b^* \leftarrow f_2(p, y; \rho_2)$	$\rho_2 \xleftarrow{r} \mathsf{Coins}_2$
6.	if $b = b^*$ output 1	

We use $[m]$ to denote the set $\{1, \ldots, m\}$ 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^λ	Random Variables
1.		$\langle a, t, b\rangle \xleftarrow{r} R_{p_0,x_0,y_0}$
2.	$b^* \leftarrow f_2(p_0, y_0; \rho_2)$	$\rho_2 \xleftarrow{r} \mathsf{Coins}_2$
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 10.3.2. 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 $\rho \xleftarrow{r} R$ 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 $\langle \rho_1, \rho_2\rangle \xleftarrow{r} R_1 \times R_2$ is one of the random variables drawn in some step of G so that ρ_1, ρ_2 are independent and ρ_2 is never used in G. G' is obtained by G by substituting $\langle \rho_1, \rho_2\rangle \xleftarrow{r} R_1 \times R_2$ with $\rho_1 \xleftarrow{r} R_1$.
4. Suppose a random variable ρ is drawn at Step i, but not used until Step j with $j > i$. Then ρ can be drawn at any Step $1, \ldots, j$.

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 10.3.1. *If G and G' are joinable,* $\mathsf{Prob}[T] = \mathsf{Prob}[T']$.

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^λ ($[p_0, x_0, c_0, y_0]$ are constants)	Random Variables
1. $b^* \leftarrow f_2(p_0, y_0; \rho_2)$	$\rho_2 \xleftarrow{r} \mathsf{Coins}_2$
2. if $b = b^*$ output 1	$b \xleftarrow{r} \{0,1\}$

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 ρ' and select it differently from the distribution of ρ. 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 ρ. The random variables ρ and $\mathsf{norm}(\rho)$ are assumed to be identically distributed. The exact formulation of the lemma is as follows.

Lemma 10.3.1. (Second Game-Playing Lemma). *Consider a game G that contains the step*

$$y \leftarrow f(y_1, \ldots, y_d, \rho_1, \ldots, \rho_i); \quad \rho_i \xleftarrow{r} R_i^{y_1,\ldots,y_d}.$$

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

$$y' \leftarrow f'(y_1, \ldots, y_d, \rho_1, \ldots, \rho_{i-1}, \rho_i'); \quad t \leftarrow \mathsf{norm}(y_1, \ldots, y_d, \rho_1, \ldots, \rho_{i-1}, \rho_i'); \quad \rho_i' \xleftarrow{r} (R_i^{y_1,\ldots,y_d})'.$$

The two games proceed identically after the ith step with every reference of y in game G substituted with y' in game G', and similarly for ρ_i substituted with t.[9] *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 ε, where ε is a function in λ (independent of the choice of K).*

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

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

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

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

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

[9] It may not be necessary to substitute ρ_i with t. For example, we omit the instruction $t \leftarrow (y_1, \ldots, y_d, \rho_1, \ldots, \rho_{i-1}, \rho_i')$ from the modified game when ρ_i does not occur in the game, when ρ_i and ρ_i' are identically distributed, and when $\rho_i' = \langle \rho_i, s \rangle$ for any s.

and

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

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

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

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

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

For all $y_0 \in D_y \cup D_{y'}$, it holds that $0 \leq C(y_0) \leq 1$. Using Lemma 10.1.1, we can then conclude

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

Now

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

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^λ	Random Variables
1.	$p \leftarrow \mathsf{PRIME}(1^\lambda, \rho)$	$\rho \xleftarrow{r} \mathsf{Coins}$
2.	$x \leftarrow b + at \bmod p$	$a \xleftarrow{r} [p],\ t \xleftarrow{r} [p-1],\ b \xleftarrow{r} \{0,1\}$
3.	$c \leftarrow f_1(p, x; \rho_1)$	$\rho_1 \xleftarrow{r} \mathsf{Coins}_1$
4.	$y \leftarrow Y$	$Y \xleftarrow{r} [p]$
5.	$b^* \leftarrow f_2(p, y; \rho_2)$	$\rho_2 \xleftarrow{r} \mathsf{Coins}_2$
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^λ ($[p_0, x_0, c_0, y_0]$ are constants)	Random Variables
1.	$\langle a, t, b\rangle \xleftarrow{r} R_{p_0,x_0,y_0}$
2. $b^* \leftarrow f_2(p_0, y_0; \rho_2)$	$\rho_2 \xleftarrow{r} \mathsf{Coins}_2$
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 $|\mathsf{Prob}[T_1] - \mathsf{Prob}[T_4]| \leq 2^{-\lambda+1}$.

10.4 The Third Game-Playing Lemma

Lemma 10.4.1. ***(Third Game-Playing Lemma)****. Let $\mathcal{D}_1^\lambda$ nd $\mathcal{D}_2^\lambda$ be two probability ensembles with $\lambda \in \mathbb{N}$, such that for all PPT $\mathcal{A}$,*

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

If G_1 and G_2 are two otherwise identical games such that whenever a sequence of variables $\vec{y} = \langle y_1, \ldots, y_d\rangle$ 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 $|\mathsf{Prob}[T_1] - \mathsf{Prob}[T_2]| \leq \delta(\lambda)$.

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 $\vec{y}$. ■

Notation. For any PPT $\mathcal{A}$ with input of size λ, 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.

11 PRPs versus PRFs

Here we analyze the pseudorandom permutation/pseudorandom function (PRP/PRF) lemma discussed in [3]. 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 $\mathcal{A}$,

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

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

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 λ. 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 $\{0,1\}^\lambda \longrightarrow \{0,1\}^\lambda$ on input 1^λ. 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^λ	Random Variables
1.	$\pi \leftarrow \mathsf{PERM}(1^\lambda; \rho)$	$\rho \xleftarrow{r} \mathsf{Coins}$
2.	$b \leftarrow \mathcal{A}^\pi(1^\lambda; \rho')$	$\rho' \xleftarrow{r} \{0,1\}^{t(\lambda)}$
3.	output b	

The winning probability of game G_1 equals $\mathsf{Prob}[\mathcal{A}^\pi(1^\lambda) = 1]$. Suppose $\mathcal{A}_0, \ldots, \mathcal{A}_Q$ 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^λ	Random Variables
1.	$\pi \leftarrow \mathsf{PERM}(1^\lambda, \rho)$	$\rho \xleftarrow{r} \mathsf{Coins}$
2.	$\langle \mathsf{aux}, i_1 \rangle \leftarrow \mathcal{A}_0(1^\lambda; \rho_1)$	$\rho_1 \xleftarrow{r} \{0,1\}^\lambda$
3.	for $j = 1$ to $Q - 1$	
4.	$\quad \mathsf{a}_j \leftarrow \pi(i_j)$	
5.	$\quad \langle \mathsf{aux}, i_j \rangle \leftarrow \mathcal{A}_j(1^\lambda, \mathsf{aux}, \mathsf{a}_j; \rho_j)$	$\rho_j \xleftarrow{r} \{0,1\}^\lambda$
6.	$\mathsf{a}_Q \leftarrow \pi(i_Q)$	
7.	$b \leftarrow \mathcal{A}_Q(1^\lambda, \mathsf{aux}, \mathsf{a}_Q; \rho_Q)$	$\rho_Q \xleftarrow{r} \{0,1\}^\lambda$
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^λ	Random Variables
1.	$\langle \mathsf{aux}, i_1 \rangle \leftarrow \mathcal{A}_0(1^\lambda; \rho_1)$	$\rho_1 \xleftarrow{r} \{0,1\}^\lambda$
2.	for $j = 1$ to $Q - 1$	
3.	$\quad \mathsf{a}_j \leftarrow v_j$	$v_j \xleftarrow{r} R_\lambda^{\mathsf{a}_1 \cdots \mathsf{a}_{j-1}}$
4.	$\quad \langle \mathsf{aux}, i_{j+1} \rangle \leftarrow \mathcal{A}_j(1^\lambda, \mathsf{aux}, \mathsf{a}_j; \rho_j)$	$\rho_j \xleftarrow{r} \{0,1\}^\lambda$
5.	$\mathsf{a}_Q \leftarrow v$	$v \xleftarrow{r} R^{\mathsf{a}_1, \ldots, \mathsf{a}_{Q-1}}$
6.	$b \leftarrow \mathcal{A}_Q(1^\lambda, \mathsf{aux}, \mathsf{a}_Q; \rho_Q)$	$\rho_Q \xleftarrow{r} \{0,1\}^\lambda$
7.	output b	

Note that $R_\lambda^{\mathsf{a}_1 \cdots \mathsf{a}_k} = \{0,1\}^\lambda - \{\mathsf{a}_1, \ldots, \mathsf{a}_k\}$

Modification [3,j]. We modify the 3rd step during the jth 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.	$\quad \mathsf{a}_j \leftarrow v_j$	$v_j \xleftarrow{r} \{0,1\}^\lambda$

When conditioning on any choice of pairwise distinct $\mathsf{a}_1, \ldots, \mathsf{a}_{j-1}$; any $i_1, \ldots, i_j$; and any $\mathsf{aux}_1, \ldots, \mathsf{aux}_j$, it holds that the probability distribution of a_j in game $G_{3,j}$ is uniform over $\{0,1\}^\lambda$. On the other hand, the probability distribution of a_j in game $G_{3,j-1}$ is uniform over $\{0,1\}^\lambda - \{\mathsf{a}_1, \ldots, \mathsf{a}_{j-1}\}$, 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 $|\mathsf{Prob}[T_{3,j-1}] - \mathsf{Prob}[T_{3,j}]| \leq (j-1)/2^\lambda$. From this, we can deduce

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

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

	Game $G_{3,Q}$ on Input 1^λ	Random Variables
1.	$\langle \mathsf{aux}, i_1 \rangle \leftarrow \mathcal{A}_0(1^\lambda; \rho_1)$	$\rho_1 \xleftarrow{r} \{0,1\}^\lambda$
2.	for $j = 1$ to $Q-1$	
3.	$\quad \mathsf{a}_j \leftarrow v_j$	$v_j \xleftarrow{r} \{0,1\}^\lambda$
4.	$\quad \langle \mathsf{aux}, i_{j+1} \rangle \leftarrow \mathcal{A}_j(1^\lambda, \mathsf{aux}, \mathsf{a}_j; \rho_j)$	$\rho_j \xleftarrow{r} \{0,1\}^\lambda$
5.	$\mathsf{a}_Q \leftarrow v$	$v \xleftarrow{r} \{0,1\}^\lambda$
6.	$b \leftarrow \mathcal{A}_Q(1^\lambda, \mathsf{aux}, \mathsf{a}_Q; \rho_Q)$	$\rho_Q \xleftarrow{r} \{0,1\}^\lambda$
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^λ	Random Variables
1.	$f \leftarrow \mathsf{FUNC}(1^\lambda; \rho)$	$\rho \xleftarrow{r} \mathsf{Coins}$
2.	$b \leftarrow \mathcal{A}^f(1^\lambda; \rho')$	$\rho' \xleftarrow{r} \{0,1\}^{t(\lambda)}$
3.	output b	

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

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

which is negligible in λ since Q is assumed to be polynomial in λ.

12 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 12.2.1. 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.

12.1 Step 1: Proving IND-CPA Security

Let GGen check be a group generator that when given a coin toss ρ and length 1^λ, it produces $\langle p, q, g_1 \rangle$ such that p is a λ-bit prime, q is an $s(\lambda)$-bit prime where $s\colon \mathbb{N} \longrightarrow \mathbb{N}$ 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 λ.

Assumption. When given a random instance of $\langle \mathbb{G}, q, g_1 \rangle$ from $\mathsf{GGen}(1^\lambda)$, the advantage $\mathsf{Adv}_{\mathrm{DDH}}^{\mathsf{GGen}}$ of any PPT $\mathcal{A}$ is negligible in λ. 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 $x, y, z \xleftarrow{r} [q]$. This is the Decisional Diffie-Hellman assumption.

12.1.1 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^λ, select a λ-bit prime p such that g_1 is a prime order q element[10], where q divides $p-1$ and q is $s(\lambda) < \lambda$ bits. Select two random values $w, z \xleftarrow{r} [q]$. 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.

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 $r \xleftarrow{r} [q]$, 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^λ	Random Variables
1.	$\langle \mathbb{G}, q, g_1\rangle \leftarrow \mathsf{GGen}(1^\lambda; \rho)$	$\rho \leftarrow \mathsf{Coins}$
2.	$g_2 \leftarrow g_1^w$	$w \xleftarrow{r} [q]$
3.	$h \leftarrow g_1^{z_1} g_2^{z_2}$	$z_1, z_2 \xleftarrow{r} [q]$
4.	$\langle \mathsf{aux}, M_0, M_1\rangle \leftarrow \mathcal{A}(\mathsf{stage}_1, g_1, g_2, h; \rho_1)$	$\rho_1 \xleftarrow{r} \mathsf{Coins}_1$
5.	$u_1^* \leftarrow g_1^r;\ \ u_2^* \leftarrow g_2^r$	$r \xleftarrow{r} [q]$
6.	$v^* \leftarrow h^r M_b$	$b \xleftarrow{r} \{0,1\}$
7.	$b^* \leftarrow \mathcal{A}(\mathsf{stage}_2, \mathsf{aux}, u_1^*, u_2^*, v^*; \rho_2)$	$\rho_2 \xleftarrow{r} \mathsf{Coins}_2$
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.	$v^* \leftarrow (u_1^*)^{z_1} (u_2^*)^{z_2} M_b$	$b \xleftarrow{r} \{0,1\}$

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 $\mathsf{Prob}[T_0] = \mathsf{Prob}[T_1]$.

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

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

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 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 $|\mathsf{Prob}[T_1] - \mathsf{Prob}[T_2]| \le \mathsf{Adv}^{\mathsf{GGen}}_{\mathrm{DDH}}(\lambda)$.

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

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

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

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

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 $|\mathsf{Prob}[T_2] - \mathsf{Prob}[T_3]| \le 2^{-s(\lambda)+1}$.

[10] g_1 is an element of a finite group of prime order q.

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

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

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

Suppose we condition on a choice of $p, q, g_1, g_2, h, \mathsf{aux}, M_0, M_1$; 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 α, α' 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 $|\mathsf{Prob}[T_3] - \mathsf{Prob}[T_4]| \leq 2^{-s(\lambda)+1}$.

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

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

Condition on a fixed choice of all variables leading up to Step 6; that is, take K to be an event that fixes $\mathbb{G}, q, g_1, g_2, h, \mathsf{aux}, M_0, M_1, u_1^*, u_2^*$. 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.

$$\begin{bmatrix} 1 & w \\ r & wr' \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} \log_{g_1} h \\ \log_{g_1}(\alpha / M_b) \end{bmatrix} \tag{13}$$

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 (13) 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 (13). One choice is for $b = 0$, and the other for $b = 1$. Nevertheless, neither conditioned game employs z_1, z_2 through $\overline{z} = z_1 + wz_2$ in any step before Step 6, and of course, $\overline{z}$ 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 $\mathsf{Prob}[T_4] = \mathsf{Prob}[T_5]$.

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

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

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

Theorem 12.1.1. *The two-generator ElGamal public-key cryptosystem satisfies IND-CPA security under the Decisional Diffie-Hellman assumption.*

12.2 Step 2: The IND-CCA1 Version, "Lunch-Time Attacks"

As before, we will use GGen as the group generator that takes coin tosses ρ and length 1^λ to produce p, q and g_1, where p is a λ-bit prime, q is an $s(\lambda)$-bit prime for a predetermined function $s\colon \mathbb{N} \longrightarrow \mathbb{N}$, 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 λ. We will employ the Decisional Diffie-Hellman assumption as defined in Section 12.1.

12.2.1 The CCA1-CS Public-Key Cryptosystem

We now present a $\mathsf{CCA1}$-variant of the Cramer-Shoup public-key encryption scheme.

Key-generation. For input 1^λ, select a λ-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, $w, z, x_1, x_2 \xleftarrow{\mathtt{r}} [q]$. 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 $r \xleftarrow{\mathtt{r}} [q]$ 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, $\mathsf{CCA1}$ stands for the chosen-ciphertext attack with one round of decryption queries. An $\mathsf{IND\text{-}CCA1}$ 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^λ	Random Variables
1.	$\langle p, q, g_1\rangle \leftarrow \mathsf{GGen}(1^\lambda; \rho)$	$\rho \leftarrow \mathsf{Coins}$
2.	$g_2 \leftarrow g_1^w$	$w \xleftarrow{\mathtt{r}} [q]$
3.	$c \leftarrow g_1^{x_1} g_2^{x_2}$	
4.	$h \leftarrow g_1^{z_1} g_2^{z_2}$	$z_1, z_2 \xleftarrow{\mathtt{r}} [q]$
5.	$\langle \mathsf{aux}, M_0, M_1\rangle \leftarrow \mathcal{A}^{\mathsf{Dec}(\cdot)[Q]}(\mathsf{stage}_1, g_1, g_2, c, h; \rho_1)$	$\rho_1 \xleftarrow{\mathtt{r}} \mathsf{Coins}_1$
6.	$u_1^* \leftarrow g_1^r;\ \ u_2^* \leftarrow g_2^r$	$r \xleftarrow{\mathtt{r}} [q]$
7.	$v^* \leftarrow h^r M_b;\ \ e^* \leftarrow c^r$	$b \xleftarrow{\mathtt{r}} \{0,1\}$
8.	$b^* \leftarrow \mathcal{A}(\mathsf{stage}_2, u_1^*, u_2^*, v^*, e^*; \rho_2)$	$\rho_2 \xleftarrow{\mathtt{r}} \mathsf{Coins}_2$
9.	if $b = b^*$ return 1 else return 0	

During Step 5, the decryption oracle $\mathsf{Dec}(\cdot)$ 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.	$v^* \leftarrow (u_1^*)^{z_1} (u_2^*)^{z_2} M_b;\ \ e^* \leftarrow (u_1^*)^{x_1} (u_2^*)^{x_2}$	$b \xleftarrow{\mathtt{r}} \{0,1\}$

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 $\mathsf{Prob}[T_0] = \mathsf{Prob}[T_1]$.

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

6.	$u_1^* \leftarrow g_1^r;\ \ u_2^* \leftarrow g_2^{r'}$	$r, r' \xleftarrow{\mathtt{r}} [q]$

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 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, $|\mathsf{Prob}[T_1] - \mathsf{Prob}[T_2]| \leq \mathsf{Adv}^{\mathsf{GGen}}_{\mathrm{DDH}}(\lambda)$.

Modification 3. Modify game G_2 to obtain game G_3 by

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

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, $|\mathsf{Prob}[T_2] - \mathsf{Prob}[T_3]| \leq 2^{-s(\lambda)+1}$.

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

> Set the variables
> $$\overline{z} \leftarrow z_1 + wz_2;\ \overline{x} \leftarrow x_1 + wx_2$$
> Given a ciphertext $\langle u_1, u_2, v, e\rangle$, test if $u_2 = u_1^w$ and $e = u_1^{\overline{x}}$. If both tests pass, return the value $v/u_1^{\overline{z}}$; 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 ith 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 $e = u_1^{\overline{x}} = u_1^{x_1}u_2^{x_2}$, so the ciphertext also passes the test of game $G_{4,i-1}$. The answer of game $G_{4,i}$ is then $v/u_1^{\overline{z}}$, 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 ith 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 $e \neq u_1^{\overline{x}}$. Note that if $u_2 = u_1^w$, it would mandate that $e = u_1^{\overline{x}}$. 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 $|\mathsf{Prob}[T_{4,i-1}] - \mathsf{Prob}[T_{4,i}]| \leq \mathsf{Prob}[F]$. We have now found a bound on $\mathsf{Prob}[F]$. Let us condition on a fixed choice of p, q, g_1, g_2, c and suppose F happens. Then in the ith 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

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

This follows from the fact that all queries made by the adversary prior to posing its ith query are answered by employing $\overline{x}$ 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 $\mathsf{Prob}[F] \leq 2^{-s(\lambda)+1}$.

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

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

It is easy to see that conditioning on $p, q, g_1, g_2, c, h, \mathsf{aux}, M_0, M_1$ 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 $p, q, g_1, g_2, c, h, \mathsf{aux}, M_0, M_1$ 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 α, α' 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 $|\mathsf{Prob}[T_{4,Q}] - \mathsf{Prob}[T_5]| \leq 2^{-s(\lambda)+1}$.

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

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

Consider an event K that fixes $p, q, g_1, g_2, h, \mathsf{aux}, M_0, M_1, u_1^*, u_2^*$. 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:

$$\begin{bmatrix} 1 & w \\ r & wr' \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} \log_{g_1} h \\ \log_{g_1}(\alpha/M_b) \end{bmatrix}. \tag{14}$$

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 (14). 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 (14) 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 (14), 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 $\overline{z} = z_1 + wz_2$ after Step 7, and of course, $\overline{z}$ 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 $\mathsf{Prob}[T_5] = \mathsf{Prob}[T_6]$.

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

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

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

Theorem 12.2.1. *The CCA1-CS public-key cryptosystem satisfies IND-CCA1 (lunch-time attack) security under the decisional Diffie-Hellman assumption.*

12.3 Step 3: The IND-CCA2 Version

Let GGen be a group generator that takes coin tosses ρ and length 1^λ and produces p, q and g_1 such that p is a λ-bit prime, q is an $s(\lambda)$-bit prime for a predetermined function $s\colon \mathbb{N} \longrightarrow \mathbb{N}$, 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 λ.

Assumption 1. Our first assumption is the decisional Diffie-Hellman assumption as defined in Section 12.1.

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

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

is negligible in λ. 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 $\mathsf{GGen}(1^\lambda)$, $\mathcal{H}$ from $\mathsf{UOWHF}(p,q,g_1)$, and x from $\langle g_1\rangle \times \langle g_1\rangle \times \langle g_1\rangle$. The string $\mathrm{desc}\mathcal{H}$ is a circuit describing the function $\mathcal{H}$.

12.3.1 The CS Public-Key Cryptosystem

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

Key-generation. For input 1^λ, select a λ-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, $w,z,x_1,x_2,y_1,y_2 \xleftarrow{\mathtt{r}} [q]$. Sample a universal one-way hash function $\mathcal{H}$ from $\mathsf{UOWHF}(1^\lambda)\colon \langle g_1\rangle \times \langle g_1\rangle \times \langle g_1\rangle \longrightarrow \mathbb{Z}_q$. The public key is $\langle p,q,g_1,g_2,h,c,d,\mathrm{desc}\mathcal{H}\rangle$ where $g_2 \leftarrow g_1^w, h \leftarrow g_1^{z_1}g_2^{z_2}, c \leftarrow g_1^{x_1}g_2^{x_2}, d \leftarrow g_1^{y_1}g_2^{y_2}$. 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 $r \xleftarrow{\mathtt{r}} [q]$, and returns $\langle u_1,u_2,v,e\rangle = \langle g_1^r, g_2^r, h^rM, c^rd^{\mathsf{h}r}\rangle$ where $\mathsf{h} \leftarrow \mathcal{H}(u_1,u_2,v)$.

Decryption. When given a ciphertext $\langle u_1,u_2,v,e\rangle$, the decryption function computes $\mathsf{h} \leftarrow \mathcal{H}(u_1,u_2,v)$ and tests if $e = u_1^{x_1+y_2\mathsf{h}}u_2^{x_2+y_2\mathsf{h}}$. 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^λ	Random variables
1.	$\langle p,q,g_1\rangle \leftarrow \mathsf{GGen}(1^\lambda;\rho)$	$\rho \leftarrow \mathsf{Coins}$
2.	$\mathcal{H} \leftarrow \mathsf{UOWHF}(1^\lambda,\rho')$	$\rho' \leftarrow \mathsf{Coins}'$
3.	$g_2 \leftarrow g_1^w$	$w \xleftarrow{\mathtt{r}} [q]$
4.	$c \leftarrow g_1^{x_1}g_2^{x_2};\ \ d \leftarrow g_1^{y_1}g_2^{y_2}$	$x_1,x_2,y_1,y_2 \xleftarrow{\mathtt{r}} [q]$
5.	$h \leftarrow g_1^{z_1}g_2^{z_2}$	$z_1,z_2 \xleftarrow{\mathtt{r}} [q]$
6.	$\langle \mathsf{aux}, M_0, M_1\rangle \leftarrow \mathcal{A}^{\mathsf{Dec}(\cdot)[Q_1]}(\mathsf{stage}_1, g_1,g_2,c,d,h;\rho_1)$	$\rho_1 \xleftarrow{\mathtt{r}} \mathsf{Coins}_1$
7.	$u_1^* \leftarrow g_1^r;\ \ u_2^* \leftarrow g_2^r$	$r \xleftarrow{\mathtt{r}} [q]$
8.	$v^* \leftarrow h^rM_b;\ \ e^* \leftarrow c^rd^{r\mathcal{H}(u_1^*,u_2^*,v^*)}$	$b \xleftarrow{\mathtt{r}} \{0,1\}$
9.	$b^* \leftarrow \mathcal{A}^{\mathsf{Dec}'(\cdot)[Q_2]}(\mathsf{stage}_2, u_1^*,u_2^*,v^*,e^*;\rho_2)$	$\rho_2 \xleftarrow{\mathtt{r}} \mathsf{Coins}_2$
10.	if $b = b^*$ return 1 else return 0	

The decryption oracle $\mathsf{Dec}(\cdot)$ is queried Q_1 times during Step 6 and $\mathsf{Dec}'(\cdot)$ is queried Q_2 times in Step 9. The operation of $\mathsf{Dec}(\cdot)$ is defined below.

> Given a ciphertext $\langle u_1,u_2,v,e\rangle$, test if $e = u_1^{x_1+y_1\mathsf{h}}u_2^{x_1+y_2\mathsf{h}}$ where $\mathsf{h} = \mathcal{H}(u_1,u_2,v)$. If the test passes, return the value $v/u_1^{z_1}u_2^{z_2}$; otherwise return $\perp$.

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. $v^* \leftarrow (u_1^*)^{z_1}(u_2^*)^{z_2}M_b;\ \ e^* \leftarrow (u_1^*)^{x_1+y_1\mathsf{h}}(u_2^*)^{x_2+y_2\mathsf{h}};$ $\mathsf{h} \leftarrow \mathcal{H}(u_1^*,u_2^*,v^*)$	$b \xleftarrow{\mathtt{r}} \{0,1\}$

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, $\mathsf{Prob}[T_0] = \mathsf{Prob}[T_1]$.

Modification 2. Perform the following modification on game G_1 to obtain game G_2:

7. $u_1^* \leftarrow g_1^r;\ u_2^* \leftarrow g_2^{r'}$	$r, r' \xleftarrow{\mathtt{r}} [q]$

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 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 $|\mathsf{Prob}[T_1] - \mathsf{Prob}[T_2]| \le \mathsf{Adv}^{\mathsf{GGen}}_{\mathrm{DDH}}(\lambda)$.

Modification 3. Modify game G_2 as follows to obtain game G_3:

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

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

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

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 $|\mathsf{Prob}[T_2] - \mathsf{Prob}[T_3]| \le 2^{-s(\lambda)+1}$.

Modification [4,*i*] where $i = 1, \ldots, Q_1$. Modify the answer to the ith query to the decryption oracle in Step 6.

> Set the variables
>
> $$\overline{z} \leftarrow z_1 + wz_2;\ \overline{x} \leftarrow x_1 + wx_2;\ \overline{y} \leftarrow y_1 + wy_2$$
>
> Given a ciphertext $\langle u_1, u_2, v, e\rangle$, test if $u_2 = u_1^w$ and $e = u_1^{\overline{x}+\overline{y}\mathsf{h}}$, where $\mathsf{h} = \mathcal{H}(u_1, u_2, v)$.
> If both tests pass, return the value $v/u_1^{\overline{z}}$; 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 ith 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 $e = u_1^{\overline{x}+\overline{y}\mathsf{h}} = u_1^{x_1+y_1\mathsf{h}}u_2^{x_2+y_2\mathsf{h}}$; that is, the ciphertext also passes the test of game $G_{4,i-1}$. The answer of game $G_{4,i}$ equals $v/u_1^{\overline{z}}$, 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 ith 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 $e = u_1^{x_1+y_1\mathsf{h}}u_2^{x_2+y_2\mathsf{h}}$, while either $u_2 \ne u_1^w$ or $e \ne u_1^{\overline{x}+\overline{y}\mathsf{h}}$. Note that we cannot have $u_2 = u_1^w$ since this implies $e = u_1^{\overline{x}+\overline{y}\mathsf{h}}$. If the event F occurs, we can assume $u_2 \ne u_1^w$ 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

$$|\mathsf{Prob}[T_{4,i-1}] - \mathsf{Prob}[T_{4,i}]| \le \mathsf{Prob}[F].$$

Thus we have found a bound on $\mathsf{Prob}[F]$. Fix a choice of p, q, g_1, g_2, c, d and suppose F occurs. Then in the ith query, $\mathcal{A}$ outputs u_1, u_2, e with $\log_{g_1} u_1 \ne \log_{g_2} u_2$ and $e = u_1^{x_1+y_1\mathsf{h}}u_2^{x_2+y_2\mathsf{h}}$.

Observe now that the random coin tosses x_1, x_2, y_1, y_2, in the view of $\mathcal{A}$ up until the ith query, must satisfy the first two equations of

$$\begin{bmatrix} 1 & w & 0 & 0 \\ 0 & 0 & 1 & w \\ \log_{g_1} u_1 & w \log_{g_2} u_2 & \mathsf{h} \log_{g_1} u_1 & w\mathsf{h} \log_{g_2} u_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} \overline{x} \\ \overline{y} \\ \log_{g_1} e \end{bmatrix} \tag{15}$$

This follows from the fact that all queries made by the adversary prior to posing its ith query are answered using $\overline{x}$ and $\overline{y}$, not x_1, x_2, y_1, y_2. The ith query itself forms the third equation of (15).

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 $\mathsf{Prob}[F] \leq 2^{-s(\lambda)+1}$.

Modification 5. Continuing with our notation from Modification 4 in Section 12.1, modify $G_4 = G_{4,Q_2}$ to obtain game G_5.

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

If we condition on $p, q, g_1, \mathcal{H}, g_2, c, d, h, \mathsf{aux}, M_0, M_1$, 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 $p, q, g_1, \mathcal{H}, g_2, c, d, h, \mathsf{aux}, M_0, M_1$; 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 α, α' with $\log_{g_1} \alpha \neq \log_{g_2} \alpha'$. Then the the conditioned games $G_4[K \wedge K_4]$ and $G_5[K \wedge K_5]$ are equivalent. By the second game playing lemma, $|\mathsf{Prob}[T_4] - \mathsf{Prob}[T_5]| \leq 2^{-s(\lambda)+1}$.

Modification 6. Modify the operation of the decryption oracle $\mathsf{Dec}'(\cdot)$ in Step 9 to form G_6.

Set the variables

$$\overline{z} \leftarrow z_1 + wz_2;\ \overline{x} \leftarrow x_1 + wx_2;\ \overline{y} \leftarrow y_1 + wy_2.$$

Given a ciphertext $\langle u_1, u_2, v, e \rangle$, test if $u_2 = u_1^w$ and $e = u_1^{\overline{x} + \overline{y}\mathsf{h}}$, where $\mathsf{h} = \mathcal{H}(u_1, u_2, v)$. If both tests pass, return $v / u_1^{\overline{z}}$; 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, $|\mathsf{Prob}[T_6] - \mathsf{Prob}[T_5]| \leq \mathsf{Prob}[F]$ 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 $e = u_1^{x_1 + y_1\mathsf{h}} u_2^{x_2 + y_2\mathsf{h}}$, but $\log_{g_1} u_1 \neq \log_{g_2} u_2$ by the same reasoning as in $G_{4,i}$. Define 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 $\neg\mathsf{Coll} \cap F$. Note that $F = F_1 \cup \ldots \cup F_{Q_2}$, where F_i denotes the event that the adversary produces a ciphertext in its ith query for which the condition of F holds. Using this, we will first place a bound on the probability $\neg\mathsf{Coll} \cap \mathsf{Prob}[F_i]$, and then apply the union bound to obtain an upper bound for the probability $\neg\mathsf{Coll} \cap F$. Let us condition over $p, q, g_1, \mathcal{H}, g_2, c, d, h, \mathsf{aux}, M_0, M_1, u_1^*, u_2^*, v^*, e^*$. Suppose that prior to the adversary making his ith query in 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.

$$\begin{bmatrix} 1 & w & 0 & 0 \\ 0 & 0 & 1 & w \\ r & wr' & \mathsf{h}^* r & w\mathsf{h}^* r' \\ \log_{g_1} u_1 & w \log_{g_2} u_2 & \mathsf{h} \log_{g_1} u_1 & w\mathsf{h} \log_{g_2} u_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} \overline{x} \\ \overline{y} \\ \log_{g_1} e^* \\ \log_{g_1} e \end{bmatrix}.$$

The adversary's ith query yields the fourth equation in the above system. The determinant is $w^2(r - r')(\mathsf{h} - \mathsf{h}^*)(\log_{g_1} u_1 - \log_{g_2} u_2)$, which is nonzero since we assume $\neg\mathsf{Coll}$. Before his ith

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 $\mathsf{Prob}[F_i \cap \neg\mathsf{Coll}] \leq 1/q$. In general, $\mathsf{Prob}[F_i \cap \neg\mathsf{Coll}] \leq 2^{-s(\lambda)+1}$. It follows then that

$$\mathsf{Prob}[\neg\mathsf{Coll} \cap F] \leq Q_2 \cdot 2^{-s(\lambda)+1}.$$

Finally, using the fact that

$$\mathsf{Prob}[F] = \mathsf{Prob}[\neg\mathsf{Coll} \cap F] + \mathsf{Prob}[\mathsf{Coll} \cap F] \leq \mathsf{Prob}[\neg\mathsf{Coll} \cap F] + \mathsf{Prob}[\mathsf{Coll}],$$

we obtain the result

$$|\mathsf{Prob}[T_6] - \mathsf{Prob}[T_5]| \leq Q_2 \cdot 2^{-s(\lambda)+1} + \mathsf{Prob}[\mathsf{Coll}].$$

We still need a bound on $\mathsf{Prob}[\mathsf{Coll}]$, which we will find in the next game.

Modification 7. Alter Step 8 of game G_6 to obtain the game G_7.

8. $v^* \leftarrow g_1^{r_3}$; $e^* \leftarrow (u_1^*)^{x_1}(u_2^*)^{x_2}$	$b \xleftarrow{\mathtt{r}} \{0,1\}$, $r_3 \xleftarrow{\mathtt{r}} [q]$

Let us condition on a fixed choice of all variables prior to Step 7: i.e., we consider an event K that fixes $p, q, g_1, \mathcal{H}, g_2, h, \mathsf{aux}, M_0, M_1, u_1^*, u_2^*$. 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 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 $\overline{z} = z_1 + wz_2$ after Step 8. The variable $\overline{z}$ 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, $\mathsf{Prob}[T_6] = \mathsf{Prob}[T_7]$.

Now let Coll_6 be the event Coll as defined in Modification 6: Coll_6 is the event that the adversary produces a collision for $\mathcal{H}$ in the second stage. Define a similar event 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 $\mathsf{Prob}[T_6'] = \mathsf{Prob}[\mathsf{Coll}_6]$ and $\mathsf{Prob}[T_7'] = \mathsf{Prob}[\mathsf{Coll}_7]$. By the same arguments used in the transition from G_6 to G_7, we find $\mathsf{Prob}[\mathsf{Coll}_6] = \mathsf{Prob}[\mathsf{Coll}_7]$.

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

$$|\mathsf{Prob}[T_7'] - \mathsf{Prob}[T_8']| = |\mathsf{Prob}[\mathsf{Coll}_7] - \mathsf{Prob}[\mathsf{Coll}_8]| \leq 2^{-s(\lambda)+1}$$

since this is the statistical distance between the two distributions.

We can now bound $\mathsf{Prob}[\mathsf{Coll}_8]$. 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

$$\mathsf{Prob}[\mathsf{Coll}_8] \leq \mathsf{Adv}^{\mathsf{GGen}}_{\mathsf{UOWHF}}(\lambda).$$

Closing argument. From the above, we can see that $\mathsf{Prob}[T_7] = 1/2$; moreover, we proved $\mathsf{Prob}[\mathsf{Coll}] \leq 2^{-s(\lambda)+1} + \mathsf{Adv}^{\mathsf{GGen}}_{\mathsf{UOWHF}}(\lambda)$ through the sequence of games G_6', G_7', G_8'. Based on this, our sequence of games allows us to conclude

$$\left|\mathsf{Prob}[T_0] - \frac{1}{2}\right| \leq \mathsf{Adv}^{\mathsf{GGen}}_{\mathrm{DDH}}(\lambda) + \mathsf{Adv}^{\mathsf{GGen}}_{\mathsf{UOWHF}}(\lambda) + (Q_1 + Q_2 + 3) \cdot 2^{-s(\lambda)+1}.$$

This is negligible in λ since Q_1, Q_2 are polynomial bounded in λ and $\mathsf{Adv}^{\mathsf{GGen}}_{\mathrm{DDH}}(\lambda)$ and $\mathsf{Adv}^{\mathsf{GGen}}_{\mathsf{UOWHF}}(\lambda)$ are assumed to be negligible in λ (by Assumptions 1 and 2). This proves the following theorem.

Theorem 12.3.1. *The CS public-key cryptosystem satisfies IND-CCA2 security under Assumptions 1 and 2.*

13 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.

13.1 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 7.6 that a message M is signed by $H(M)^d = \sigma$ and a message-signature pair (M, σ) is verified by testing if $\sigma^e \equiv H(M) \bmod n$. We now adapt this for a blind signature.

Definition 13.1.1. 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 $r \xleftarrow{r} \mathbb{Z}_n$, 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 σ' of a blinded message M', compute $\sigma = \sigma' r^{-1} \bmod n$.

- The verification algorithm Verify: For any (M, σ), 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 22 describes the general formulations of these algorithms for a user $\mathcal{U}$ and a signer $\mathcal{S}$.

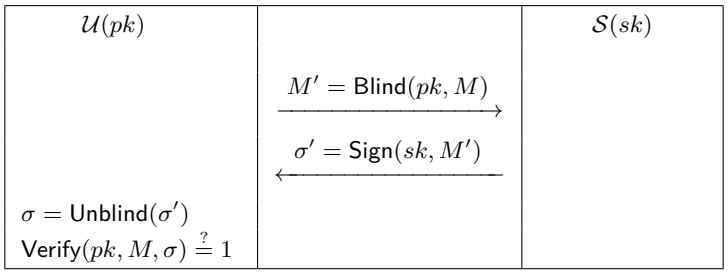


Figure 22: The Chaum blind signature generation.

Definition 13.1.2. The ***blindness property*** states that for any two messages M_0, M_1 and all algorithms $\mathcal{A}$

$$|\mathsf{Prob}[\mathcal{A}(\mathsf{Blind}(pk, M_0)) = 1] - \mathsf{Prob}[\mathcal{A}(\mathsf{Blind}(pk, M_1)) = 1]| \leq \varepsilon.$$

When ε is negligible, we say there is ***statistical blindness***; when $\varepsilon = 0$, there is ***perfect blindness***, i.e. the distributions of $\mathsf{Blind}(pk, M_0)$ and $\mathsf{Blind}(pk, M_1)$ are statistically or perfectly indistinguishable.

Theorem 13.1.1. *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 $\gcd(n, e) = 1$. ■

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 $r \xleftarrow{\mathtt{r}} \mathbb{Z}_n$.
4. $\mathcal{U}$ calculates $y = r^e H(M \| \mathtt{rnd}) \bmod n$ 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 $\mathsf{coin} = r^{-1}\sigma \bmod n$ and outputs $\langle \mathtt{rnd}, \mathsf{coin} \rangle$ 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 $\langle \mathtt{rnd}, \mathsf{coin} \rangle$ to $\mathcal{M}$. Before providing the purchased services, $\mathcal{M}$ checks that $(\mathsf{coin})^e \equiv H(M \| \mathtt{rnd}) \bmod n$. If it is valid, $\mathcal{M}$ submits $\langle \mathtt{rnd}, \mathsf{coin} \rangle$ to the bank through an authenticated channel.

When the bank receives $\langle \mathtt{rnd}, \mathsf{coin} \rangle$, 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 σ_i' for v_i'. To cast his ballot, V_i retrieves the unblinded signature σ_i of v_i from σ_i'. He verifies that (v_i, σ_i) is a valid ballot-signature pair using the administrator's verification key. If Verify returns `True`, V_i sends (v_i, σ_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, σ_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.

13.2 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 pubic key mechanism to eliminate private channels between the mix-server and the senders. To achieve this, we use the ElGamal encryption scheme:

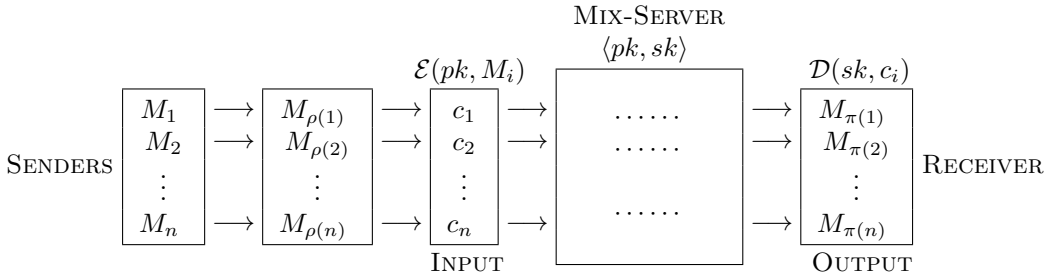


Figure 23: A mix-server with random permutations ρ, π on $\{1, \ldots, n\}$.

$\mathcal{G}(1^\lambda)$: $\langle pk, sk \rangle \leftarrow \mathcal{G}(1^\lambda)$
$x \xleftarrow{r} \mathbb{Z}_m$, $h = g^x \bmod p$
$pk = \langle \langle p, m, g \rangle, h \rangle$
$sk = x$

$\mathcal{E}(pk, M)$: $M \in \langle g \rangle$
$r \xleftarrow{r} \mathbb{Z}_m$
compute $G = g^r \bmod p$, $H = h^r M \bmod p$
output $\langle G, H \rangle$

$\mathcal{D}(sk, g, H)$: compute $M = H/G^x \bmod p$
output M

Figure 23 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 ρ 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 24.

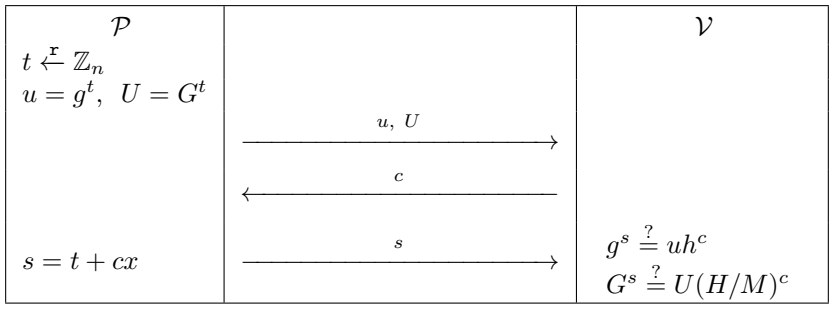


Figure 24: A zero-knowledge proof that verifies if $x = \log_g h = \log_G H/M$.

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 π 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

$$\mathsf{L} = \left\{ \langle c_1, \ldots, c_n, M_1', \ldots, M_n' \rangle \colon \text{ there is } (\pi, sk) \text{ such that } \mathcal{D}(sk, c_i) = M_{\pi(i)}' \text{ for all } 1 \le i \le n \right\}.$$

L is in NP. Given π 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

$$\mathsf{L}' = \left\{ \langle c_1, \ldots, c_n, M_1', \ldots, M_n' \rangle \colon \text{ there is an } sk \text{ such that for all } i \text{ there is a } j \text{ with } \mathcal{D}(sk, c_i) = M_j' \right\}$$

This language can also be written as

$$\mathsf{L}' = \bigwedge_{i=1}^{n} \left(\bigvee_{j=1}^{n} \mathcal{D}(sk, c_i) = m_j \right).$$

Note that $\mathsf{L} \subsetneq \mathsf{L}'$ since L' does not require that all original messages appear, i.e., in L', a mixer could set $\mathcal{D}(sk, c_i) = M_j'$ for all i and a fixed j. 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 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. If 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 $2, 3, \ldots, n$. 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 $\langle G, H\rangle = \langle g^r, (h_1h_2)^r M\rangle$ and outputs $\langle G', H'\rangle = \langle Gg^{r'}, h_2^{r'}H/G^{x_1}\rangle$. 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.

14 Distributing Trust

14.1 Secret sharing

In a ***secret sharing scheme***, a set of players posses pieces of information that combine to form a common "secret". Unless the number of involved players exceeds a certain threshold, noone 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 $i = 1, \ldots, n$. The dealer randomly selects $n-1$ numbers $s_1, \ldots, s_{n-1} \xleftarrow{r} \mathbb{Z}_q$ and sets the nth as $s_n = S - s_1 - \ldots - s_{n-1} \bmod q$. 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 $s_1 \xleftarrow{r} \mathbb{Z}_q$ 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.

14.2 Shamir's Secret Sharing Scheme

Define $p(X) = a_0 + a_1X + \cdots + a_{t-1}X^{t-1}$ for $a_i \in \mathbb{Z}_q$. If we know t points $(z_0, y_0), (z_1, y_1), \ldots, (z_{t-1}, y_{t-1})$, where $y_i = p(z_i)$, we can build the system

$$\begin{bmatrix} 1 & z_0 & \cdots & z_0^{t-1} \\ 1 & z_1 & \cdots & z_1^{t-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & z_{t-1} & \cdots & z_{t-1}^{t-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{t-1} \end{bmatrix} \equiv \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{t-1} \end{bmatrix} \bmod q. \tag{16}$$

Denote (16) by $ZA \equiv Y \bmod q$. 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: $a_1, \ldots a_{t-1} \xleftarrow{r} \mathbb{Z}_q$. D distributes $s_i = p(i)$ to each shareholder P_i for $i = 1, \ldots, n$.

When t players meet, they can solve a system similar to (16). Denote this subset of players as $\{P'_j\}$ for $j = 1, \ldots, t$, where each of the t contributing players has the corresponding share s'_j. Using Lagrange interpolation, they can obtain $a_0 = \lambda_1 s'_1 + \ldots + \lambda_t s'_t$, where each λ_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.

14.3 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 14.2. Distribute $s_i = p(i)$ to the ith shareholder P_i for $i = 1, \ldots, n$.

When a group of t players P'_j, $j = 1, \ldots, t$, decide to decrypt $\langle G, H\rangle$, each P'_j publishes $G_j = G^{s'_j}$.

P'_1	P'_2	$\cdots$	P'_t
s'_1	s'_2	$\cdots$	s'_t
G_1	G_2	$\cdots$	G_t

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 14.3.1. Given two groups $(X, +)$ and $(Y, \cdot)$[11], a ***group homomorphism*** is a function $\varphi\colon X \longrightarrow Y$ such that for all $\alpha, \beta \in X$,

$$\varphi(\alpha + \beta) = \varphi(\alpha) \cdot \varphi(\beta).$$

This implies φ must also preserve identities and inverses.

Define $(X, +)$ to be the plaintext group and $(Y, \cdot)$ to be the cipertext 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

$$\mathcal{E}(pk, M_1 + M_2) \approx C_1 \cdot C_2$$[12].

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 $\{h^{M_1}, h^{M_2}, \ldots, h^{M_m}\}$. 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 ith 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 $i = 1, \ldots, n$. 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.

[11] 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.

[12] The symbol $\approx$ represents an identical distribution

14.4 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

$$\langle g^{a_0}, g^{a_1}, \ldots, g^{a_{t-1}} \rangle = \langle V_0, V_1, \ldots, V_{t-1} \rangle.$$

If s_i corresponds to the ith shareholder, notice

$$\begin{aligned} V_0 V_1^i \cdots V_{t-1}^{i^{t-1}} &= g^{a_0} g^{a_1 i} \cdots g^{a_{t-1} i^{t-1}} \\ &= g^{p(i)} \\ &= g^{s_i}, \end{aligned}$$

so the ith 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 ith 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 ith player then provides a zero-knowledge proof for the discrete logarithms, as discussed in previous sections.

14.5 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 $p_i(X) = a_{i,0} + a_{i,1}X + \ldots + a_{i,t-1}X^{t-1}$ to compute $S_{i,j} = p_i(j)$. He gives $S_{i,j}$ to P_j as the jth share of S_i. P_i then publishes $V_{i,0}, V_{i,1}, \ldots, V_{i,t-1}$.

The jth shareholder now collects $S_{1,j}, S_{2,j}, \ldots, S_{n,j}$ 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 $k = 0, \ldots, t-1$. Note that no non-sharing entity knows the secret, but the secret can still be recovered by any attempt of t users.

15 Broadcast Encryption

In a system with users in $[N] = \{1, \ldots, N\}$, 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 $\mathcal{D}(k, c)$.

After the network initializes the system with $\mathsf{Init}(key)$, 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 $C = \langle \mathcal{E}(k_{E_1}, M), \ldots, \mathcal{E}(k_{E_{N-r}}, M) \rangle$ 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.

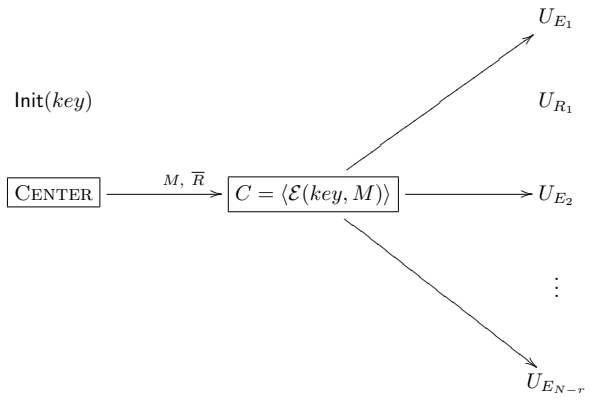


Figure 25: A broadcast encryption scheme where users U_{E_i} can decrypt M, but U_{R_j} cannot, with $E_i \in E$ and $R_i \in R$. $\overline{R}$ denotes the description of R.

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 $[N]$[13]. 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 26. 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 26, 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.

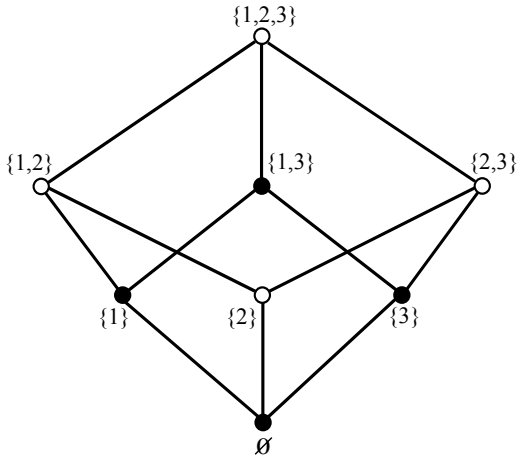


Figure 26: All possible subsets of $[N]$ for $N = 3$ and $R = \{2\}$.

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 $\lambda = |\mathcal{E}(key, M)|$. 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.

[13]The term poset comes from "partially ordered set"

15.1 Complete Binary Trees

One effective way to minimize both the ciphertext size and user memory space is to use complete binary trees.

Definition 15.1.1. A ***complete binary tree*** (CBT) is a tree diagram in which every level is full and all levels have the same depth[14]. Any CBT has 2^t leaves for some $t \in \mathbb{Z}^+$.

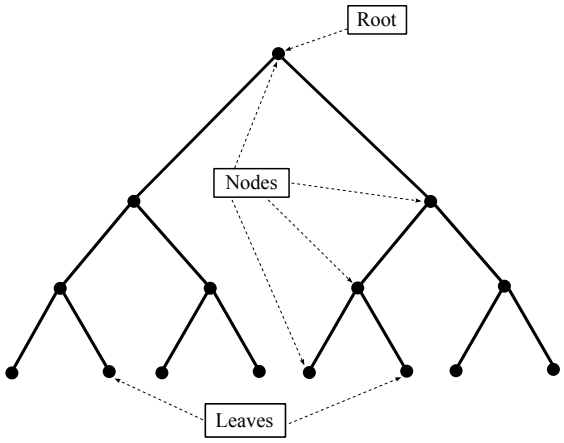


Figure 27: A complete binary tree with $N = 8$ leaves and depth 3.

Lemma 15.1.1. *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 28 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.

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 29, where open nodes denote excluded keys.

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.

16 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.

[14]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.

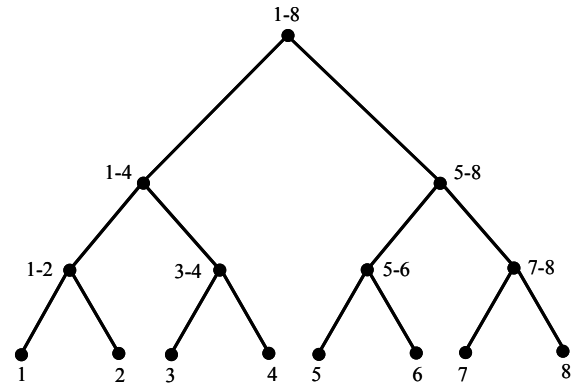


Figure 28: A complete binary tree representing $N = 8$ users in a broadcast encryption scheme.

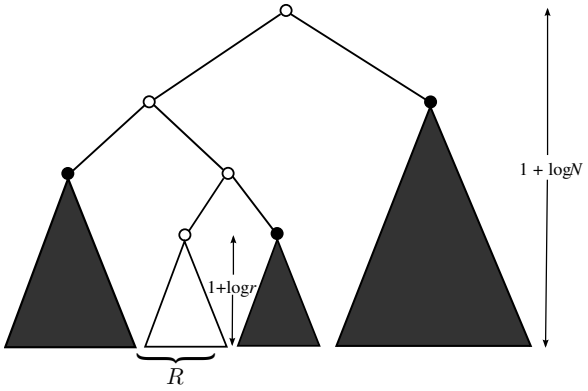


Figure 29: Excluding R when R forms a single complete subtree.

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 know 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.

16.1 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[15] 2 or 3, we can write any elliptic curve as a plane algebraic curve defined by[16]

$$Y^2 = X^3 + aX + b, \tag{17}$$

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.

[15]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.

[16]Equation (17) 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 ∞. 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

$$x_3 = m^2 - x_1 - x_2 \qquad \text{and} \qquad y_3 = y_1 + m(x_3 - x_1),$$

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.

16.2 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 16.2.1. Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two groups of prime order q[17]. An admissible ***bilinear map***

$$e \colon \mathbb{G}_1 \times \mathbb{G}_1 \longrightarrow \mathbb{G}_2$$

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 paring function, we can prove several complexity implications [6].

Theorem 16.2.1. *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, α and β both have prime order q, so this establishes a reduction of the discrete logarithm problem into $\mathbb{G}_2$. ∎

Theorem 16.2.2. *The decisional Diffie-Hellman problem is easy in $\mathbb{G}_1$.*

[17] 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.

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 $a, b, c \xleftarrow{r} \mathbb{Z}_q^*$ 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 $c = ab \bmod q$, $g_1 = g_2$. The distinguisher can therefore tell the tuples apart. ∎

16.3 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 16.3.1. 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 ε in solving BDH if

$$\mathsf{Adv}^{\mathcal{A}} = \mathsf{Prob}[\mathcal{A}(P, aP, bP, cP) = e(P, P)^{abc}] \geq \varepsilon.$$

The hardness of BDH depends on an appropriate choice of security parameters in $\mathbb{G}_2$ [**?**].

Definition 16.3.2. A ***BDH parameter generator*** is a randomized, polynomial-time computable algorithm $\mathcal{G}$ that on input λ, 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 $e\colon \mathbb{G}_1 \times \mathbb{G}_1 \longrightarrow \mathbb{G}_2$. We denote the output of $\mathcal{G}$ on the security parameter λ as $\mathcal{G}(1^\lambda) = \langle q, \mathbb{G}_1, \mathbb{G}_2, e\rangle$.

Definition 16.3.3. A BDH parameter generator $\mathcal{G}$ is said to satisfy the ***bilinear Diffie-Hellman assumption*** provided

$$\mathsf{Adv}_{\mathcal{G}}^{\mathcal{A}}(\lambda) = \mathsf{Prob}\left[\mathcal{A}(q, \mathbb{G}_1, \mathbb{G}_2, e, P, aP, bP, cP) = e(P, P)^{abc} : \begin{array}{l}\langle q, \mathbb{G}_1, \mathbb{G}_2, e\rangle \leftarrow \mathcal{G}(1^\lambda), \\ P \xleftarrow{r} \mathbb{G}_1^*, a, b, c \xleftarrow{r} \mathbb{Z}_q^*\end{array}\right]$$

is negligible for all PPT algorithms $\mathcal{A}$. Informally, the BDH assumption considers it hard to compute $e(P, P)^{abc}$ when given $\langle P, aP, bP, cP\rangle$.

16.4 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

16.5 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 $\widehat{e}$ 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 16.5.1.

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^λ, run $\mathcal{G}$ to obtain the system parameters $\langle q, \mathbb{G}_1, \mathbb{G}_2, \widehat{e}\rangle$ where q is a λ-bit prime, $\mathbb{G}_1$ and $\mathbb{G}_2$ are groups of order q, and $\widehat{e}$ is an admissible bilinear map $\widehat{e}\colon \mathbb{G}_1 \times \mathbb{G}_1 \longrightarrow \mathbb{G}_2$. Choose a random generator P of $\mathbb{G}_1$. Set $s \xleftarrow{r} \mathbb{Z}_q^*$ as the system-wide secret key and $P_{pub} = sP$ as the system-wide public key. Choose $Q_{id} \xleftarrow{r} \mathbb{G}_1^*$. Let H_2 be a cryptographic hash function $H_2\colon \mathbb{G}_2 \longrightarrow \{0,1\}^n$ for some n. The public key is $\langle q, \mathbb{G}_1, \mathbb{G}_2, \widehat{e}, n, P, P_{pub}, Q_{id}, H_2\rangle$. The private key is $d_{id} = sQ_{id} \in \mathbb{G}_1^*$.

Encryption. Given a message $M \in \{0,1\}^n$, choose $r \xleftarrow{r} \mathbb{Z}_q^*$ and return $C = \langle rP, M \oplus H_2(g^r)\rangle$, where $g = \widehat{e}(Q_{id}, P_{pub}) \in \mathbb{G}_2^*$.

Decryption. Given a ciphertext $C = \langle U, V\rangle$, use d_{id} to compute $V \oplus H_2(\widehat{e}(d_{id}, U)) = M$.

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 16.5.1. 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 $a, b, c, d \xleftarrow{r} \mathbb{Z}_q^*$.

Proof of Security. As in Section 12.1.1, we can model the security as a sequence of games.

The IND-CPA Game.

	Game G_0 on Input 1^λ	Random Variables
1.	$\langle q, \mathbb{G}_1, \mathbb{G}_2, \widehat{e}\rangle \leftarrow \mathcal{G}(1^\lambda; \rho)$	$\rho \leftarrow \mathsf{Coins}$
2.	$P_{pub} \leftarrow sP$	$s \xleftarrow{r} \mathbb{Z}_q^*$
3.	$Q_{id} \xleftarrow{r} \mathbb{G}_1^*$	
4.	$\langle \mathsf{aux}, M_0, M_1\rangle \leftarrow \mathcal{A}^{H_2(\cdot)[q_{H_2}]}(\mathsf{stage}_1, q, \mathbb{G}_1, \mathbb{G}_2, \widehat{e}, n, P, P_{pub}, Q_{id}, H_2; \rho_1)$	$\rho_1 \xleftarrow{r} \mathsf{Coins}_1$
5.	$U^* \leftarrow rP$	$r \xleftarrow{r} \mathbb{Z}_q^*$
6.	$V^* \leftarrow M_b \oplus H_2(\widehat{e}(Q_{id}, P_{pub})^r)$	$b \xleftarrow{r} \{0,1\}$
7.	$b^* \leftarrow \mathcal{A}(\mathsf{stage}_2, \mathsf{aux}, U^*, V^*; \rho_2)$	$\rho_2 \xleftarrow{r} \mathsf{Coins}_2$
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.	$Q_{id} \leftarrow tP$	$t \xleftarrow{r} \mathbb{Z}_q^*$

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 $\mathsf{Prob}[T_0] = \mathsf{Prob}[T_1]$.

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 $i = 1, \ldots, q_{H_2}$, 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 $H_i \xleftarrow{r} \{0,1\}^n$, 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 $\mathsf{Prob}[T_1] = \mathsf{Prob}[T_2]$.

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 $|\mathsf{Prob}[T_2] - \mathsf{Prob}[T_3]| \leq \mathsf{Adv}^{\mathcal{A}}_{\mathrm{BDDH}}(\lambda)$.

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 $\widehat{e}(Q_{id}, P_{pub})^r \in List_{H_2}$. Clearly, if $\mathcal{A}$ does not make a query about $\widehat{e}(Q_{id}, P_{pub})^r$, $\mathsf{Prob}[T_3 \cap \neg F] = \mathsf{Prob}[T_4 \cap \neg F]$. Assuming $\#\mathbb{G}_1 \geq 2^\lambda$, the first game-playing lemma yields

$$|\mathsf{Prob}[T_3] - \mathsf{Prob}[T_4]| \leq \frac{q_{H_2}}{2^\lambda}.$$

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

$$\left|\mathsf{Prob}[T_0] - \frac{1}{2}\right| \leq \mathsf{Adv}^{\mathcal{A}}_{\mathrm{BDDH}}(\lambda) + \frac{q_{H_2}}{2^\lambda}.$$

Since both $\mathsf{Adv}^{\mathcal{A}}_{\mathrm{BDDH}}(\lambda)$ and $\frac{q_{H_2}}{2^\lambda}$ are assumed to be negligible in λ, this is also negligible. This proves the following theorem.

Theorem 16.5.1. *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 $G_{\mathsf{IND\text{-}ID\text{-}CPA}}$ on Input 1^λ	Random Variables
1.	$\langle \mathsf{param}, sk_A \rangle \leftarrow \mathcal{G}(1^\lambda; \rho)$	$\rho \xleftarrow{r} \{0,1\}^\lambda$
2.	$\langle \mathsf{aux}, id, M_0, M_1 \rangle \leftarrow \mathcal{A}^{K(sk_A,\cdot)}(\mathsf{stage}_1, \mathsf{aux}, \mathsf{param}; \rho_1)$	$\rho_1 \xleftarrow{r} \{0,1\}^\lambda$
3.	if $id \in Computed$ or $M_0 = M_1$ then stop	
4.	$C \leftarrow \mathcal{E}(\mathsf{param}, id, M_b)$	$b \xleftarrow{r} \{0,1\}$
5.	$b^* \leftarrow \mathcal{A}^{K^{\neg id}(sk_A,\cdot)}(\mathsf{stage}_2, \mathsf{aux}; \rho_2)$	$\rho_2 \xleftarrow{r} \{0,1\}^\lambda$
11.	if $b = b^*$ return 1 else return 0	

We now shift our attention to a stronger IBE system, BasicIdent:

Setup. For input 1^λ, run $\mathcal{G}$ to obtain the system parameters $\langle q, \mathbb{G}_1, \mathbb{G}_2, \widehat{e} \rangle$ where q is a λ-bit prime, $\mathbb{G}_1$ and $\mathbb{G}_2$ are groups of order q, and $\widehat{e}$ is an admissible bilinear map $\widehat{e}\colon \mathbb{G}_1 \times \mathbb{G}_1 \longrightarrow \mathbb{G}_2$. Choose a random generator P of $\mathbb{G}_1$. Set $s \xleftarrow{r} \mathbb{Z}_q^*$ 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 $H_1\colon \{0,1\}^* \longrightarrow \mathbb{G}_1^*$ and $H_2\colon \mathbb{G}_2 \longrightarrow \{0,1\}^n$ 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 $r \xleftarrow{r} \mathbb{Z}_q^*$, 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 16.5.2. *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 $G_{\mathsf{IND\text{-}ID\text{-}CCA}}$ on Input 1^λ	Random Variables
1.	$\langle \mathsf{param}, sk_A \rangle \leftarrow \mathcal{G}(1^\lambda; \rho)$	$\rho \xleftarrow{r} \{0,1\}^\lambda$
2.	$\langle \mathsf{aux}, id, M_0, M_1 \rangle \leftarrow \mathcal{A}^{K(sk_A,\cdot),\mathsf{Dec}(K(sk_A,\cdot),\cdot)}(\mathsf{stage}_1, \mathsf{aux}, \mathsf{param}; \rho_1)$	$\rho_1 \xleftarrow{r} \{0,1\}^\lambda$
3.	if $id \in Computed$ or $M_0 = M_1$ then stop	
4.	$C \leftarrow \mathcal{E}(\mathsf{param}, id, M_b)$	$b \xleftarrow{r} \{0,1\}$
5.	$b^* \leftarrow \mathcal{A}^{K^{\neg id}(sk_A,\cdot),\mathsf{Dec}^{\neg(id,C)}(K(sk_A,\cdot),\cdot)}(\mathsf{stage}_2, \mathsf{aux}; \rho_2)$	$\rho_2 \xleftarrow{r} \{0,1\}^\lambda$
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 16.5.2. A public key identity based encryption scheme is IND-ID-CCA secure if for all PPT adversaries $\mathcal{A}$,

$$\mathsf{Adv}^{\mathcal{A}}_{\mathsf{IND\text{-}ID\text{-}CCA}}(\lambda) = \left| \mathsf{Prob}[b = b^*] - \frac{1}{2} \right|$$

is negligible.

17 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 [5].

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 $P_1, P_2, \ldots$ 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 30 $\mathcal{A}$ corrupts P_2, so now P_2 shares all of his information with $\mathcal{A}$.

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 $(\mathsf{InitiatorStart}, sid)$, I initiates a key exchange protocol with a party R, whose name is determined by sid.

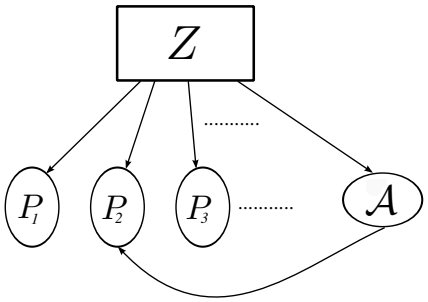


Figure 30: An environment Z with users P_i and adversary $\mathcal{A}$.

- When the responding party R is contacted by the initiating party, it produces the output $(\mathsf{ResponderAck}, sid)$ in acknowledgment.
- When the responder R receives input $(\mathsf{ResponderStart}, sid)$, 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 $(\mathsf{InitiatorStart}, sid)$ 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[18] $(\mathsf{ResponderAck}, sid)$ to R.
- Upon receiving $(\mathsf{ResponderStart}, sid)$ from R, verify R is recorded as the responder, record R as active, and then notify the adversary.
- After receiving a message $(\mathsf{Key}, sid, P, k^*)$ 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.

17.1 The 2DH Key Exchange Protocol

To exhibit the differences between the real and ideal worlds, recall the Diffie-Hellman key exchange protocol from Section 6.1. Here we refer to this as the 2DH key exchange protocol. In

[18] 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.

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 31, the protocol now operates as follows.

1. Z sends $(\mathsf{InitiatorStart}, (I, R))$ 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 $(\mathsf{ResponderAck}, (I, R))$ to Z to acknowledge he was contacted by I.
5. Z sends $(\mathsf{ResponderStart}, (I, R))$ 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.

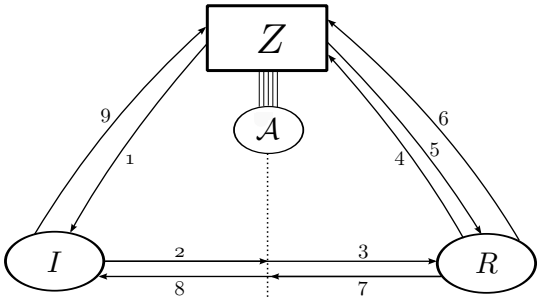


Figure 31: The 2DH key exchange protocol in the real world simulation.

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 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 32, the 2DH key-exchange protocol in the ideal world is as follows.

1. Z sends $(\mathsf{InitiatorStart}, (I, R))$ 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 $(\mathsf{ResponderAck}, (I, R))$ to Z to acknowledge he was contacted by $\mathcal{F}_{KE}(I)$
5. Z sends $(\mathsf{ResponderStart}, (I, R))$ 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.

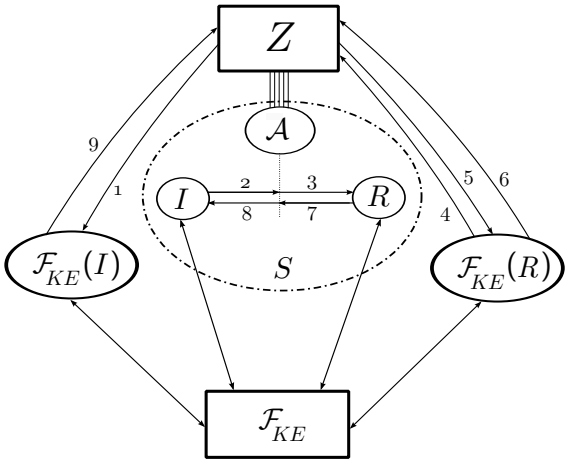


Figure 32: The 2DH key-exchange protocol in the ideal world simulation.

Definition 17.1.1. 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.

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 6.4 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 6.4 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.

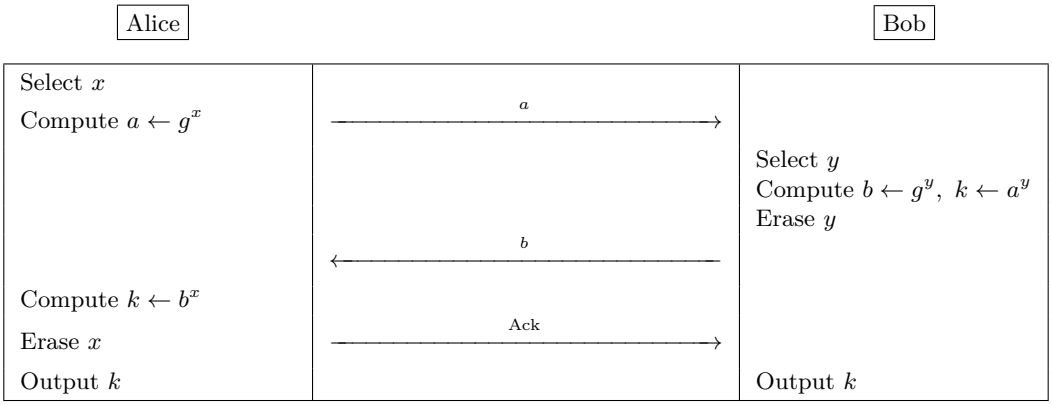


Figure 33: The 2mDH key exchange protocol.

17.2 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 33.

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 indistinguishablility necessary between the real and ideal worlds.

18 Private Information Retrieval

Assume that there is a database DB $= \langle x_1, ..., x_n \rangle$, stored in a remote server (or servers). Consider the setting where a user U makes a query q_i to retrieve an item x_i from DB. A *Private information Retrieval (PIR)* protocol, is a protocol where the queries U makes are formed so that the correct item is retrieved without revealing the position of x_i in DB to the (potentially malicious) server.

18.1 Information Theoretic PIR

The notion was introduced in 1995 by Chor, Goldreich, Kushilevitz and Sudan in the information theoretic setting. An (information theoretic) PIR protocol must satisfy the following properties:

1. **Correctness:** when U makes query q_i, he receives x_i from the server.
2. **Privacy:** Let Q be a *query function* that takes as input an index in $[n]$ and the user's random coins r and let $\mathcal{Q}$ be the query space. Then, for every distinct indices $i, j \in [n]$ and $q \in \mathcal{Q}$:
$$\underset{r}{\mathsf{Prob}}[Q(i,r) = q] = \underset{r}{\mathsf{Prob}}[Q(j,r) = q].$$

Namely, the queries of the user hide the position of the corresponding item against unbounded adversaries. A trivial solution to the problem is to ask the server to send the whole database to the user. Indeed, if U receives DB, then he can easily extract x_i, while the server cannot distinguish in which of all the items U is interested. However, this is a very inefficient protocol, as the communication and user-side storage overhead are unbearable for large databases. Therefore, the interesting solutions are the ones that allow the server's response to be *sublinear* to the size of the database. Chor, Goldreich, Kushilevitz and Sudan have proven that no sublinear PIR protocol exists in the information theoretic setting, as long as the database is stored in a *single server*.

18.2 Computational PIR

In 1997, Kushilevitz and Ostrovsky showed that single-server PIR is possible in the computational setting, by relaxing the privacy definition. A *computational PIR* (CPIR) protocol is private if the queries for different items are indistinguishable by a polynomial time bounded adversary . Formally, for every PPT adversary $\mathcal{A}$, database size n polynomial to the security parameter λ and distinct indices $i, j \in [n]$:

$$| \operatorname*{Prob}_r[\mathcal{A}(1^\lambda, Q(i,r)) = 1] - \operatorname*{Prob}_r[\mathcal{A}(1^\lambda, Q(j,r)) = 1] | = \mathsf{negl}(\lambda).$$

We will present a sublinear CPIR protocol for the case where each item is a single bit, i.e. $x_i \in \{0,1\}$ for $i = 1, ..., n$. We view the database as a $s \times t$ matrix of bits, where $n = s \cdot t$. Assume a mapping from indices in $[n]$ to matrix entries, e.g. $i \longmapsto (a,b) = (\lceil i/t \rceil, i \bmod t)$. We will need the following notion:

Definition 18.2.1. (**XOR-homomorphic encryption**) An asymmetric encryption scheme $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ where the message space $\mathcal{M}$ is $\{0,1\}$ is *XOR-homomorphic* if for any public key pk and $M, M' \in \mathcal{M}$:

$$\mathcal{E}(pk, M) \cdot \mathcal{E}(pk, M') = \mathcal{E}(M \oplus M').$$

Let $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ be an XOR-homomorphic scheme. The CPIR protocol works as follows:

1. U generates a public key and secret key pair $\langle pk, sk \rangle$ for $(\mathcal{G}, \mathcal{E}, \mathcal{D})$. He creates ciphertexts $y_1, \ldots, y_t$, such that y_b is an encryption of 0, while for all $j \neq b$, y_j is an encryption of 1. U sends pk and $y_1, \ldots, y_t$ to the server.

2. Upon receiving pk and $y_1, \ldots, y_t$, the server computes for every row $\sigma \in [s]$ a number $z_\sigma \in \mathbb{Z}_N^*$ as follows: for $j = 1, \ldots, t$ it computes

$$w_{\sigma,j} = \begin{cases} y_j^2, & \text{if the item in entry } (\sigma, j) \text{ is 0} \\ y_j, & \text{if the item in entry } (\sigma, j) \text{ is 1} \end{cases}$$

   Then it computes $z_\sigma = \prod_{j=1}^{t} w_{\sigma,j}$, for $\sigma = 1, \ldots, s$ and responds with $z_1, \ldots, z_s$.

3. Using $z_1, \ldots, z_s$, U retrieves the item in entry (a,b) by deciding 0 if $\mathcal{D}(sk, z_a) = 0$ and 1 otherwise.

It is obvious that during an execution of the protocol, $(1+t)+s$ strings (the public key and $t+s$ ciphertexts are transmitted. Assume, without loss of generality, that the public key and the ciphertexts have size λ bits and $s = t = \sqrt{n}$, where $n = \lambda^c$ for some $c > 2$. Then the total communication cost is $(1+s+t) \cdot \lambda = (1 + 2 \cdot n^{\frac{1}{2}}) \cdot n^{\frac{1}{c}} = \Theta(n^{\frac{1}{2}+\frac{1}{c}}) = \omega(n)$ bits, i.e. sublinear to the size of the database. Next, we prove the two properties of the CPIR protocol.

Correctness: By the construction of $y_1, \ldots, y_t$, if $j \neq b$, then

$$w_{\sigma,j} = \begin{cases} \mathcal{E}(0) \cdot \mathcal{E}(0), & \text{if the item in entry } (\sigma, j) \text{ is 0} \\ \mathcal{E}(0), & \text{if the item in entry } (\sigma, j) \text{ is 1} \end{cases}$$

Thus, by the XOR-homomorphic property, for $j \neq b$, $w_{\sigma,j}$ is an encryption of 0 for $j \neq b$. On the other hand,

$$w_{\sigma,b} = \begin{cases} \mathcal{E}(1) \cdot \mathcal{E}(1), & \text{if the item in entry } (\sigma, b) \text{ is 0} \\ \mathcal{E}(1), & \text{if the item in entry } (\sigma, b) \text{ is 1} \end{cases}$$

Therefore, by the XOR-homomorphic property, $w_{\sigma,b}$ is an encryption of 0 if and only if the item in entry (σ, b) is 0. By the construction of z_σ and he XOR-homomorphic property, z_σ is in an encryption of 0 if and only if $w_{\sigma,b}$ is an encryption of 0. Therefore, z_σ is an encryption of 0 if and only if the item in entry (σ, b) is 0 and especially, z_a is an encryption of 0 if and only if the item in entry (a, b) is 0.

Privacy: We will show that if $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ is IND-CPA secure, then the CPIR protocol is private. Let $\mathcal{A}$ be a PPT adversary that breaks the privacy of the CPIR protocol. Namely, for two distinct entries $(a, b) \neq (a', b')$ and a non-negligible function $\alpha(\cdot)$:

$$\underset{r}{\mathsf{Prob}}[\mathcal{A}(1^\lambda, Q((a,b), r)) = 1] = \theta \quad \text{and} \quad \underset{r}{\mathsf{Prob}}[\mathcal{A}(1^\lambda, Q((a',b'), r)) = 1] \geq \theta + \alpha(\lambda).$$

We observe that the protocol works in an identical way for entries that are in the same column, therefore it must hold that $b \neq b'$. We construct an algorithm $\mathcal{B}$ against the IND-CPA security of $(\mathcal{G}, \mathcal{E}, \mathcal{D})$. $\mathcal{B}$ invokes $\mathcal{A}$ and acts as a user of the protocol as follows:

1. On input pk, $\mathcal{B}$ sets challenge messages $0, 1$ and receives a ciphertext y that is either an encryption of 0 or an encryption of 1. $\mathcal{B}$ creates $t-2$ encryptions of 0, denoted as y_j, where $j \notin \{b, b'\}$. It chooses randomly a position $\widetilde{b} \in \{b, b'\}$ and sets $y_{\widetilde{b}} = y$. Finally, for the remaining position $\widetilde{b'} \in \{b, b'\} \setminus \widetilde{b}$ it creates another encryption of 0 and sets it as $y_{\widetilde{b'}}$. It sends pk and $y_1, \ldots, y_t$ to $\mathcal{A}$.

2. If the chosen position is b', then $\mathcal{B}$ answers by flipping $\mathcal{A}$'s output. If the chosen position is b, then $\mathcal{B}$ returns $\mathcal{A}$'s output.

The cases that $\mathcal{B}$ decides correctly are when: (i) it outputs 1 and y is an encryption of 1 and (ii) it outputs 0 and y an encryption of 0.

Let E be the event that y is an encryption of 0. By definition of IND-CPA security, we have that $\mathsf{Prob}[E] = 1/2$. If E occurs, all $y_1, \ldots, y_t$ are encryptions of 0 for any choice of $\widetilde{b}$. Therefore,

$$\mathsf{Prob}[\mathcal{A}(1^\lambda, (pk, y_1, \ldots, y_t)) = 1 \mid E \wedge \widetilde{b} = b] = \mathsf{Prob}[\mathcal{A}(1^\lambda, (pk, y_1, \ldots, y_t)) = 1 \mid E \wedge \widetilde{b} = b'] = \phi.$$

So,

$$\begin{aligned}\mathsf{Prob}[\mathcal{B} = 0 \mid E] &= \mathsf{Prob}[\widetilde{b} = b] \cdot \mathsf{Prob}[\mathcal{B} = 1 \mid E \wedge \widetilde{b} = b] + \mathsf{Prob}[\widetilde{b} = b'] \cdot \mathsf{Prob}[\mathcal{B} = 1 \mid E \wedge \widetilde{b} = b'] = \\ &= 1/2 \cdot (\mathsf{Prob}[\mathcal{B} = 0 \mid E \wedge \widetilde{b} = b] + \mathsf{Prob}[\mathcal{B} = 0 \mid E \wedge \widetilde{b} = b']) = \\ &= 1/2 \cdot (\phi + (1 - \phi)) = 1/2,\end{aligned} \tag{18}$$

by the description of $\mathcal{B}$. On the other hand, if y is an encryption of 1, then a complete execution of the CPIR is simulated where the query $(pk, y_1, \ldots, y_t)$ is made for the item in entry (a, b), if $\widetilde{b} = b$ or (a', b'), if $\widetilde{b} = b'$. Therefore,

$$\begin{aligned}\mathsf{Prob}[\mathcal{B} = 1 \mid \neg E] &= \mathsf{Prob}[\widetilde{b} = b] \cdot \mathsf{Prob}[\mathcal{B} = 0 \mid \neg E \wedge \widetilde{b} = b] + \mathsf{Prob}[\widetilde{b} = b'] \cdot \mathsf{Prob}[\mathcal{B} = 0 \mid \neg E \wedge \widetilde{b} = b'] = \\ &= 1/2 \cdot (\mathsf{Prob}[\mathcal{B} = 0 \mid \neg E \wedge \widetilde{b} = b] + \mathsf{Prob}[\mathcal{B} = 0 \mid \neg E \wedge \widetilde{b} = b']) \geq \\ &\geq 1/2 \cdot ((1 - \theta) + \theta + \alpha(\lambda)) = 1/2 + \alpha(\lambda)/2.\end{aligned} \tag{19}$$

By (18),(19) we have that

$$\begin{aligned}\mathsf{Prob}[\mathcal{B} \text{ succeeds}] &= \mathsf{Prob}[\mathcal{B} = 0 \wedge E] + \mathsf{Prob}[\mathcal{B} = 1 \wedge \neg E] = \\ &= \mathsf{Prob}[E] \cdot \mathsf{Prob}[\mathcal{B} = 0 \mid E] + \mathsf{Prob}[\neg E] \cdot \mathsf{Prob}[\mathcal{B} = 1 \mid \neg E] \geq \\ &\geq 1/2 \cdot (1/2 + (1/2 + \alpha(\lambda)/2)) = 1/2 + \alpha(\lambda)/4,\end{aligned}$$

hence $\mathcal{B}$ breaks the IND-CPA security of $(\mathcal{G}, \mathcal{E}, \mathcal{D})$.

18.3 An instantiation of a XOR-homomorphic asymmetric encryption scheme.

Recall that for an integer N, the subgroup of all quadratic residues of $\mathbb{Z}_N^*$ is $QR(N)$ (see Definition 6.2.4). We will need the following statements:

Proposition 18.3.1. *Let $N = p \cdot q$, where p, q are primes. The following hold:*

(i). It is easy to sample a random element $y \in \mathbb{Z}_N^*$ that is not in $QR(N)$.

(ii). If the factorization of N is known, then it is easy to decide whether an element $y \in \mathbb{Z}_N^*$ is in $QR(N)$.

Proposition 18.3.2. *Let $N = p \cdot q$, where p, q are primes and $NQR(p, q)$ be the set of elements in $\mathbb{Z}_N^*$ that are neither in $QR(p)$ nor in $QR(q)$. Then the following hold:*

(i). $|QR(N) \cup NQR(p, q)| = |\mathbb{Z}_N^*|/2$, i.e. for exactly half of the elements $y \in \mathbb{Z}_N^*$, either $y \in QR(N)$ or $y \in NQR(p, q)$.

(ii). $|QR(N)| = |NQR(p, q)|$.

Definition 18.3.1. (**Quadratic Residuosity Assumption).** Let $N = p \cdot q$, where p, q are random λ-bit primes. For, a random element in $y \in QR(N) \cup NQR(p, q)$, the *Quadratic Residue Problem* is to decide whether y is in $QR(N)$. The *Quadratic Residuosity Assumption* states that for every PPT adversary $\mathcal{A}$, the probability that $\mathcal{A}$ solves the Quadratic Residue Problem is no more than $1/2 + \mathsf{negl}(\lambda)$.

The following asymmetric encryption scheme is XOR-homomorphic and has IND-CPA security under the Quadratic Residuosity Assumption.

- $\mathcal{G}(1^\lambda)$: generate two random λ-bit primes p, q. Set $\langle pk, sk\rangle = \langle N = p \cdot q, (p, q)\rangle$.
- $\mathcal{E}(N, M)$: choose a random y that is not in $QR(N)$. This can be done efficiently due to proposition 18.3.1.(i). If M is 0, encrypt as y^2 mod N and if M is 1 encrypt as y mod N.
- $\mathcal{D}((p, q), c)$: Decrypt to 0 if c is in $QR(N)$, otherwise decrypt to 1. The decryption is easy due to Proposition 18.3.1.(ii).

19 The bitcoin protocol

In 2008 a person or a group of people self-identified as Satoshi Nakamoto publicized the bitcoin protocol. Bitcoin is a cryptocurrency, in the sense that it makes use of cryptography and it can be used to make payments. Compared to traditional electronic payment systems, bitcoin does not have a central authority that issues money and validates transactions. Instead, trust is distributed and the different parties involved in the bitcoin system take part of this trust.

However, distributing trust comes with a number of challenges. A distributed payment system must protect its users from double spending. Since no central authority is in charge of validating transactions, a malicious user can try to spend the same amount of money twice. This problem is addressed by the bitcoin protocol by ensuring that all parties have the same view of the history of transactions.

Next, we give an outline of the bitcoin protocol. The basic element of the bitcoin protocol is a transaction. A transaction determines the flow of money: a payer transfers an amount of money to a payee. Payer and payee are identified by a randomly generated public key which makes bitcoin a pseudonymous payment system: anyone can use it without exposing any information about his identity besides a pair of randomly generated cryptographic keys. Transactions are processed in blocks. A sequence of blocks is called a chain, and ordering is ensured through the use of a hash function. Formally, let $B_1 = \langle s_1, x_1, r_1\rangle$ represent a block, where s_1 is the hash of the previous blocks, x_1 is the set of transactions in this block and r_1 is a nonce. Assuming $\mathcal{H}$ is a cryptographic hash function, for the next block $B_2 = \langle s_2, x_2, r_2\rangle$ in the chain it should hold that $s_2 = \mathcal{H}(s_1, H(x_1, r_1))$. This way for a chain containing blocks $B_1, \ldots, B_n$, the probability that some malicious entity finds some block B_i' such that $B_1, \ldots, B_i', \ldots, B_n$ is a valid chain is equal to the probability that it finds a collision in the hash function i.e. negligible.

Another novelty of bitcoin compared to older payment systems is that it is permissionless, anyone can take part in the payment infrastructure. A known attack against permisionless

systems is the sybil attack. Since these systems lack identity infrastructure, an attacker can make many 'fake' identities in order to bias decisions taken by the system. Bitcoin addresses this problem by the use of proofs-of-work. A proof-of-work is a proof that someone has done computational work at some point in the past with regard to some specific context. In bitcoin, this is captured by the requirement that the hash of a block must be less or equal to some value T i.e. $\mathcal{H}(s_1, H(x_1, r_1)) < T$ in order for $\langle s_1, x_1, r_1 \rangle$ to be a valid block. T is called the difficulty of block B_1. Difficulty is recalculated every 2016 blocks in order to ensure that one block is being produced every ten minutes. Block miners are incetivized to spend computational power by being rewarded bitcoins for each block they mine.

Finally, in order to defend against double spending, the bitcoin protocol defines a 'chain selection rule' so that all parties converge to the same transaction history. This rule specifies that all parties must select and mine on the most difficult chain that they have received or mined so far. If there is a tie, the chain that was received earlier should be selected. It has been proven that under suitable assumptions this mechanism leads to a common transaction history for all players.

19.1 The q-bounded synchronous setting

We will describe a simple model one can use to argue about the security properties of bitcoin. The model is a multiparty synchronous communication setting (similar to Canetti's formulation of "real world" execution [5]) with the relaxation that the underlying communication graph is not fully connected and messages are delivered through a "diffusion" mechanism that reflects Bitcoin's peer-to-peer structure.

Parties. Each party P is represented by an interactive Turing machine having the following communication tapes:

- *input tape*: a read-only tape from where P gets his input (for example transactions transmitted in the bitcoin network)
- *receive tape*: a read-only tape from which P receives messages
- *output tape*: a write-only tape where P writes his output
- *broadcast tape*: a write-only tape which P uses to broadcast messages

As stated before, the network is considered to be synchronous, i.e. the protocol takes place in successive rounds. Synchronicity is modeled by having the players share a read-only variable called "round". In each round, parties are able to read their input tape (Input()) and receive tape (Receive()), perform some computation that will be suitably restricted and issue a Broadcast message that is guaranteed to be delivered to all parties in the beginning of the next round.

Proofs-of-Work. In order to capture the parties' limited ability to produce proofs-of-work (POW), we assume that all parties have access to a random oracle $H(\cdot)$. We consider parties that can ask a bounded number of queries to the oracle. A random oracle is called q-bounded if each player asks at most q times the oracle a *hash* query at each round. Also notice, that if a party is trying to find a block of difficulty T, the probability of success is $T/2^{\kappa}$, where κ is the length of the output of the hash.

Environment. In order to specify an execution of a protocol Π in this system we add two more participants, the environment and the controller. The controller manages the scheduling of the activations of the participants in the execution. It also increases the round variable at the appropriate time. The environment Z provides the parties' inputs, and also receives the parties' outputs. The environment may provide input to a party at any round and may also modify that input from round to round. It is not permitted any queries to $H(\cdot)$. The rationale for this is that we would like to bound the "CPU power" of the adversary to be proportionate to the number of parties it controls, while making it infeasible for them to be aided by external sources

or by transferring the hashing power potentially invested in concurrent or previous protocol executions. It follows that in our analysis we will focus on the "standalone" setting, where a single protocol instance is executed in isolation. The execution proceeds as follows:

1. The environment receives as input the output and broadcast tapes of the players. It then writes at their input tape. If this is the first round, the environment sets the round variable to 1, otherwise it increases it by 1.

2. The controller writes the concatenation of the messages (possibly reordering them according to a probability distribution based on some network characteristics) in the broadcast tapes to the incoming communication tape of each player.

3. Each player P_1 to P_n is executed one after another. If a player writes a query to the shared tape with the oracle, the random oracle is activated in order to respond and then the player is executed again.

The adversary. The adversary is modeled by another Turing machine that takes as input the broadcast tapes of the players. The adversarial model in the network is "rushing", meaning that in any given round the adversary gets to see all honest players' messages before deciding his strategy, and, furthermore, also allows the adversary to change the source information on every message. Note that the adversary cannot change the contents of the messages nor prevent them from being delivered. Effectively, this parallels communication over TCP/IP in the Internet where messages between parties are delivered reliably, but nevertheless malicious parties may "spoof" the source of a message they transmit and make it appear as originating from an arbitrary party (including another honest party) in the view of the receiver. In this setting we use Broadcast as the message transmission command that captures the "send-to-all" functionality allowed by our communication model. Note that an adversarial sender may abuse Broadcast and attempt to confuse honest parties by sending and delivering inconsistent messages to them.

Additionally the adversary has at his disposal qt queries to the q-bounded oracle. The order of the execution is altered as follows to include the adversary:

1. The environment takes as input the output and broadcast tapes of the players and the output of the adversary. It then writes at their input tape. If this is the first round the controller sets the round variable to 1, otherwise it increases it by 1.

2. The adversary is activated next. It obtains the contents of all broadcast tapes and for each player, he writes a list of messages in the incoming communication tape in any order he wants. Each list must include all messages placed in the broadcast tapes of honest players in the previous round.

3. Each player P_1 to P_n is executed one after another. If a player writes a query to the shared tape with the random oracle, the oracle is activated in order to respond and then the player is executed again.

4. The adversary is activated next. It performs qt queries, and terminates by writing something to its output tape if it wants.

We assume that all executions are *static*, i.e. all players are spawned in the first round an no one is deleted. Further, note that because of the unauthenticated nature of the communication model the parties may never be certain about the number of participants in a protocol execution.

We refer to the above restrictions on the environment, the parties and the adversary as the *q-bounded synchronous setting*.

19.2 The core lemma

As discussed earlier honest parties try to have the same view of the history of transactions. An event that helps them achieve this goal is a uniquely successful round, i.e. a round where only one honest player succeeds in mining a block. Unless the adversary publishes some other block

of his own, a uniquely successful round forces all honest players to the same chain in the next round. Thus, it is important for the number of uniquely successful rounds to be greater than the number of blocks mined by the adversary in a fixed interval of rounds.

Let the random variable X_i be 1 if round i is uniquely successful and 0 otherwise. Assume that $Pr[X_i = 1] = \gamma$ and for $i \neq j$, X_i is independent of X_j. Let random variable $Z_{i,j}$ be 1 if the adversary succeeds in his j-th query in the i-th round. Assume $\mathbb{E}[\sum_{j=1}^{qt} Z_{i,j}] = \beta$ and without loss of generality let $\{Z_{i,j}\}_{i,j\in\mathbb{N}}$ be independent and identically distributed. We can now state the lemma we are going to prove.

Lemma 19.2.1. *Assume $\gamma > (1+\delta)\beta$, for some $\delta \in (0,1)$. Then, for any $s \in \mathbb{N}$, the probability that $\sum_{i=1}^{s} X_i$ is less than $(1+\delta/2)\sum_{i=1}^{s}\sum_{j=1}^{qt} Z_{i,j}$ is negligible in s.*

Proof. Let $X = \sum_{i=1}^{s} X_i$ and $Z = \sum_{i=1}^{s}\sum_{j=1}^{qt} Z_{i,j}$. Since $X_1, \ldots, X_s$ and $Z_{1,1}, \ldots, Z_{s,qt}$ are i.i.d Bernoulli random variables we can apply the Chernoff bound twice.

$$Pr[X \leq (1-\delta/8)\gamma s] \leq e^{-\gamma s\delta^2/128} \leq \mathsf{negl}(s) \tag{20}$$

$$Pr[Z \geq (1+\delta/9)\beta s] \leq e^{-\beta s\delta^2/243} \leq \mathsf{negl}(s) \tag{21}$$

By the union bound, the probability that any of those two events happens is also negligible in s. Hence, the following sequence of inequalities holds with probability $1 - \mathsf{negl}(s)$:

$$X > (1-\delta/8)\gamma s \geq (1-\delta/8)(1+\delta)\beta s \geq (1+\delta/2)(1+\delta/9)\beta s > (1+\delta/2)Z$$

The first and the last inequalities hold from the negation of Inequalities 20 and 21. The second inequality holds from our assumption and the third inequality follows by simple calculations. The lemma follows. ∎

For more details regarding the modeling of Bitcoin and its security we refer to [7].

20 Secure Multiparty Computation

Secure multiparty computation refers to the problem of securely computing the outcome of the $f(x_1, \ldots, x_n)$ for a public function f and inputs $x_1, \ldots, x_n$ that are contributed by n parties.

We will examine a variant of the problem where the inputs are provided by n "input-providers", the computation is performed by k processors $S_1, \ldots, S_k$ and the output is obtained by a single output receiver.

The objective is that the output receiver should learn nothing about $x_1, \ldots, x_n$ except what may be inferred by $f(x_1, \ldots, x_n)$ while any coalition of processors numbering less than k members should not be able to infer any information whatsoever about the inputs.

To present the protocol we will restrict ourselves to the setting of $x_1, \ldots, x_n \in \{0,1\}$ and a function f that can be expressed as a Boolean circuit consisting of gates XOR, AND and NOT.

20.1 The protocol

We will use arithmetic in $\mathbb{F}_2 = \{0,1\}$ which is a finite field with operations[19] $(+, \cdot)$. We will use the following encoding over $\mathbb{F}_2$.

$$[x] = \langle x^{(1)}, \ldots, x^{(k)} \rangle \text{ s.t. } \sum_{j=1}^{k} x^{(j)} = x$$

In the protocol, whenever we refer to an encoding $[x]$, the j-th processor will hold the coordinate $x^{(j)}$.

Prior to the onset of the computation the processors $S_1, \ldots, S_k$ perform a protocol to compute a number of "Beaver triples." This enables them to compute encodings $[x], [y], [z]$ for values $x, y, z \in \mathbb{F}_2$ that satisfy $z = x \cdot y$.

[19] We note that in $\mathbb{F}_2$, behaves $+$ as XOR i.e. $1 + 1 = 0$.

To start the protocol, each input provider prepares a random encoding $[x_i]$ and transmits privately the j-th component $x_i^{(j)}$ to the j-th processor.

The processors then traverse the circuit gate by gate, calculating the encoding of the output wire given the encodings of the input wires.

We consider the following three cases.

- For a NOT gate, with input encoding $[x] = \langle a^{(1)}, \ldots, a^{(k)} \rangle$, the output is set to
$$\langle 1 + a^{(1)}, a^{(2)}, \ldots, a^{(k)} \rangle$$
- For a XOR gate, with input encodings $[a] = \langle a^{(1)}, \ldots, a^{(k)} \rangle$, $[b] = \langle b^{(1)}, \ldots, b^{(k)} \rangle$, the output is set to
$$\langle a^{(1)} + b^{(1)}, \ldots, a^{(k)} + b^{(k)} \rangle$$
- For an AND gate, with input encodings $[a] = \langle a^{(1)}, \ldots, a^{(k)} \rangle$, $[b] = \langle b^{(1)}, \ldots, b^{(k)} \rangle$, each processor broadcasts to all processors the values
$$d^{(j)} = a^{(j)} - x^{(j)},\ e^{(j)} = b^{(j)} - y^{(j)}$$
where [x], [y], [z], is a Beaver triple.
Subsequently, each processors defines
$$s^{(j)} = o_j \cdot d \cdot e + d \cdot y^{(j)} + e \cdot x^{(j)} + z^{(j)}$$
where $o_j = 1$ if $j = 0$ and 0 otherwise.

Prove the following as an exercise.

Proposition 20.1.1. *The encoding $[s]$ defined above satisfies $s = a \wedge b$.*

When the traversal terminates, the output of the circuit is a single wire $[f(x_1, \ldots, x_k)]$. Each processor transmits its coordinate to the output receiver who sums the values to obtain the output of the computation.

20.2 Oblivious Transfer and Beaver Triple Computation

In this section we describe how the processors perform the precomputation step that enables them to compute the Beaver triples $[x], [y], [z]$. We will consider the special case $k = 2$.

We will utilize a primitive called Oblivious Transfer. This cryptographic primitive enables Alice, playing the role of the sender, and Bob playing the role of the receiver to engage in the following interaction. Alice holds two values x_0, x_1 and Bob a single bit $b \in \{0, 1\}$. At the end of the protocol, Bob should obtain x_b, while learning nothing about the value x_{1-b}. We will restrict ourselves to the setting that $x_0, x_1 \in \{0, 1\}$.

We will use an oblivious transfer protocol that is based on ElGamal encryption. Recall that g will be a prime order cyclical group of order q. The oblivious transfer protocol is described as follows.

1. Alice sends to Bob value $A = g^s$, where $s \leftarrow \mathbb{Z}_q$.
2. Bob responds with $B = g^r A^b$ where $r \leftarrow \mathbb{Z}_q$.
3. Alice prepares two values (Y_0, Y_1) as follows: $Y_d = (B \cdot A^{-d})^s$, where $d \in \{0, 1\}$ and sends $(C_0, C_1) = (Y_0 \cdot g^{x_0}, Y_1 \cdot g^{x_1})$ to Bob.
4. Bob calculates $V = C_b \cdot A^{-r}$ and returns 0 if $V = 1$, while 1 otherwise.

Prove as an exercise the following proposition.

Proposition 20.2.1. *(i) If both parties follow the protocol, then Bob at the end of the interaction calculates the correct value x_b. (ii) Alice learns nothing about the value b of Bob. (iii) if both parties follow the protocol, the value x_{1-b} of Alice, given the values b, r, C_0, C_1, can be guessed with probability no better than distinguishing the pair (g^s, g^{s^2}) from a random pair over $\langle g \rangle$.*

Suppose we rewrite $x_1 = x_0 + \delta$. Then, we can rewrite Bob's output x_b as $x_b = x_0 + b \cdot \delta$. That is, when operating over bits, selection is akin to multiplication. We will expand upon this on the following paragraph.

Building Beaver Triples. We are now ready to use Oblivious Transfer to construct beaver triples. Alice and Bob each select three random bits: (r_a, x_a, y_a), and (r_b, x_b, y_b). They will use Oblivious Transfer, to produce z_a, z_b so that: only Alice knows z_a, only Bob knows z_b, and additionally $[z] = [x][y]$, where $x = x_a + x_b$, $y = y_a + y_b$, $z = z_a + z_b$. We will use the above observation, and have Alice share $(r_a, r_a + x_a)$ and Bob select using bit y_b. Bob's output will then be $v_b = r_a + x_a \cdot y_b$.

Next, we have bob share $(r_b, r_b + x_b)$ and Alice select using bit y_a. Alice's output will then be $v_a = r_b + x_b \cdot y_a$. Finally, we set $z_b = v_b + r_b + x_b \cdot y_b$, and $z_a = v_a + r_a + x_a \cdot y_a$. This is possible as Alice knows (r_a, x_a, y_a) and Bob knows (r_b, x_b, y_b).

We observe that $z_a + z_b = x_a \cdot y_b + x_b \cdot y_b + x_b \cdot y_a + x_a \cdot y_a = (x_a + x_b) \cdot (y_a + y_b) = x \cdot y$.

21 Cryptographic Contact Tracing

Contact tracing is the process through which a person who is diagnosed with an infectious disease can notify people who they interacted with in a specific time window. The recent spread of COVID-19 has drawn attention to contact tracing as an effective mitigation of the spread of the disease, with a strong interest in a way to automate the process.

An automated process would require some sort of logging to take place, as notifications need to be made to people with whom the infected interacted with in the past. As such, it is only natural that safeguards are built into such a system, so as not to leak any information other than what is strictly necessary.

We will begin with describing the protocols of such a system and their general operation, before stating security requirements and definitions, or suggesting a candidate design.

High-Level Description:

A Cryptographic Contact Tracing (CCT) system consists of a central server and three protocols:

- Contact: a protocol which is run between users (or their phones), with the purpose of (localy) recording each meeting by users of the system.
- Scan: a protocol run between a user and the server, with a purpose of notifying the user if one of his past meetings was with a newly diagnosed person.
- Notify: a protocol that allows users to notify the server that they have recently been diagnosed.

As a starting assumption, we assume that the server is honest but curious, and that the notify procotol is augmented with some out of band system that verifies infection (e.g. via a registered medical professional).

21.1 Syntax of a CCT protocol

21.1.1 Contact

The contact protocol is run between users to keep track of contacts between each other.

- $\mathsf{Contact.Init}(1^\lambda) \to \mathsf{st}_0$ Initialises a new state st_0 with keysize λ

- $\mathsf{Contact.Send}(\mathsf{st}) \to (\mathsf{st}', \mathsf{msg}')$, Updates its state and produces a new message for broadcasting.
- $\mathsf{Contact.Recv}(\mathsf{st}, \mathsf{msg}) \to \mathsf{st}'$. Given a message msg from another user, the user stores it locally by updating the state.

Intended use: Alice first initialises her state, and runs update in regular intervals. At the same time she is broadcasting the last msg she has produced. When receiving a message from other users, she uses the store interface.

21.1.2 Scan

Scan is an algorithm that is run after contacting the server to obtain a reply τ. The result depends on the server's *reported state* τ.

- $\mathsf{Scan}(vk, \mathsf{st}, \tau)$: Outputs 0 or 1 depending on the server reply τ.

Intended use: When Alice wants to check if she is at risk, she asks the server for a scan file, and the sever replies with τ. Alice locally runs $\mathsf{Scan}(vk, \mathsf{st}, \tau) \to 0/1$ with 1 meaning one of her recent contacts has been diagnosed positive. The vk is made available to Alice ahead of time (during server initialization).

21.1.3 Notify

The Notify protocol used to inform the server that a user has been diagnosed as infected. It has three interfaces:

- $\mathsf{Notify.Server}(1^\lambda) \to (vk, sk, \tau_0)$. Server Initializes by creating a verification key vk, a corresponding secret key sk, and an initial stored state τ_0. The key vk is distributed to the users.
- $\mathsf{Notify.User}(\mathsf{st}) \to \mathsf{msg}$. A user notifies the server by running $\mathsf{Notify.User}(\mathsf{st})$ to produce a message msg and sends it to the server.
- $\mathsf{Notify.Server}(sk, \mathsf{msg}, \tau) \to \tau'$. The server handles user messages by updating its state based on the user message.

Intended use: Server initialises the state. Alice, upon being diagnosed positive, runs Notify.User on her state and produces a message to the server. Upon receiving Alice's message the server updates its state.

21.2 Security Definitions

Informally, we can group our security requirements under three properties: correctness, integrity and privacy. We provide a brief description of the three areas before attempting a formal treatment.

- Correctness: when operated by honest users, the protocol should produce the desired outcome.
- Integrity: a malicious user is unable to produce false positives or false negatives.
- Privacy: the participants (i.e users and the server) should not receive any information apart from what is required.

21.2.1 Correctness

We formalize the notion as follows: when Alice runs Scan, the result is one if and only if, one of her contacts has previously run Notify.

Definition 21.2.1. A CCT scheme (Contact, Notify, Scan) is **Correct** if:

$$\begin{aligned}&\Pr[(vk, sk, \tau) \leftarrow \mathsf{Notify.Server}(1^\lambda); \mathsf{st_{bob}} \leftarrow \mathsf{Contact.Init}(1^\lambda); \mathsf{st_{alice}} \leftarrow \mathsf{Contact.Init}(1^\lambda);\\ &\quad (\mathsf{msg}, \mathsf{st'_{bob}}) \leftarrow \mathsf{Contact.Send}(\mathsf{st_{bob}}); \mathsf{st_{alice}}' \leftarrow \mathsf{Contact.Recv}(\mathsf{st_{alice}}, \mathsf{msg});\\ &\quad \mathsf{msg}' \leftarrow \mathsf{Notify.User}(\mathsf{st'_{bob}}); \tau \leftarrow \mathsf{Notify.Server}(sk, \mathsf{msg}', \tau) :\\ &\quad \mathsf{Scan}(vk, st_{\mathsf{alice}}', \tau) = 1] \approx 1\end{aligned}$$

It is also possible to allow for malicious scheduling by allowing an adversary to insert additional calls between the ones required by the experiment above (e.g that would cover a protocol that misbehaves if the contact protocol between Bob and Alice is run more than once).

21.2.2 Integrity

We can partition the integrity requirement into two issues: false positives (forcing a user's scan to output 1 when they have not contacted a user who has run Notify) and false negatives (forcing a user to output 0 when they have contacted persons running notify).

Under our envisioned setup, there exist trivial attacks that produce false negatives. Malicious users may turn off, damage, or shield their devices preventing correct data exchange. Malicious servers might discard or delay processing on Notify messages.

Given that our primary application is linked with voluntary use of an application and a government-run server, we opt to focus on false positives. In the definition, we allow the adversary to communicate freely with two honest users over a period of days. We also provide a set of dummy users named "Ian", such that ian^d will be diagnosed at the end of day d. In the base game, we do not allow the adversary to interact with the Ians. We assume that users sun Scan every night. In class, we covered "one-day" security

Game $\mathsf{CCT}^{\mathcal{U},d_{max}}_{\mathsf{INT}}(\mathcal{A}, 1^\lambda)$

$(vk, sk, \tau) \leftarrow \mathsf{Notify.Server}(1^\lambda); \mathsf{st_{bob}} \leftarrow \mathsf{Contact.Init}(1^\lambda, \epsilon); \mathsf{st_{alice}} \leftarrow \mathsf{Contact.Init}(1^\lambda); d = 0;$
#Morning
$d = d + 1$
$\mathsf{st}_{\mathsf{ian}^d} \leftarrow \mathsf{Contact.Init}(1^\lambda)$
$\perp \leftarrow \mathcal{A}^{\mathsf{OUser}}()$
$\mathsf{msg}_d \leftarrow \mathsf{Notify.User}(\mathsf{st}_{\mathsf{ian}^d}); \tau \leftarrow \mathsf{Notify.Server}(sk, \mathsf{msg}_d, \tau)$
$(\tau', next) \leftarrow \mathcal{A}(\tau, \mathsf{msg}_d)$
If $(next = 1)$ & $(d < d_{max})$ Goto #Morning
If $(\mathsf{Scan}(vk, st_{\mathsf{alice}}', \tau') = 1)$ Return 1
Return 0

In the above game the adversary has access to the following oracle: OUser.

- The user Oracle, when called with a single argument argument $\mathsf{OUser}(u)$ runs $(\mathsf{st}_u, \mathsf{msg}) \leftarrow \mathsf{Contact}(\mathsf{st}_u, \epsilon)$ and returns msg. When called with two arguments $\mathsf{OUser}(u, x)$ it runs $\mathsf{st}_u \leftarrow \mathsf{Contact}(\mathsf{st}_u, x)$. In both cases, if $u \notin \mathcal{U}$ it returns $\perp$ instead, without affecting the user's state.

In the lecture, we defined one-day security for a non-predicting adversary (i.e one who cannot predict that a user will be infected). We thus define one-day non-predicting CCT integrity (INT-NP1) as follows.

Definition 21.2.2. A CCT scheme $(\mathsf{Contact}, \mathsf{Notify}, \mathsf{Scan})$ has 1-day non-predicting integrity if for every stateful PPT adversary $\mathcal{A}$:

$$\Pr[\mathsf{CCT}_{\mathsf{INT}}^{\{\mathsf{alice},\mathsf{bob}\},1}(\mathcal{A}, 1^\lambda)] \approx 0$$

If we allow more than one day to pass, we can model adversaries who use the server's message to craft messages to other users. If we allow $\mathcal{U}$ to include ian, we can model adversaries using relay or replay attacks based on high-risk individuals.

21.2.3 Privacy

Privacy in terms of such a system can be very complex: we cover different types of users (state, healthy, infected, suspect) who might or might not be contacting each other. One way to express privacy is to state it as a negative: use the system should not reveal any information other than what is required (at a base level, informing a user that a non-zero number of their recent contacts was diagnosed).

A simpler way is to explicitly describe the kind of inferences that should not be possible in the system: i.e the state should not be able to determine if Alice has met with Bob. Towards that, we will design a game where the adversary is able to contact both users and in addition, view all communication from Alice and Bob to the server.

Game $\mathsf{CCT}_{\mathsf{PRI}}^{b}(\mathcal{A}, 1^\lambda)$

$(vk, sk, \tau) \leftarrow \mathsf{Notify.Server}(1^\lambda); \mathsf{st}_{\mathsf{bob}} \leftarrow \mathsf{Contact.Init}(1^\lambda); \mathsf{st}_{\mathsf{alice}} \leftarrow \mathsf{Contact.Init}(1^\lambda, \epsilon); d = 0;$

$\perp \leftarrow \mathcal{A}^{\mathsf{OUser},\mathsf{ONotify}}();$

$\mathsf{msg}_{\mathsf{alice}} \leftarrow \mathsf{Contact.Send}(\mathsf{st}_{\mathsf{alice}})$

$\mathsf{If}\ (b = 1) : \mathsf{st}_{\mathsf{bob}} \leftarrow \mathsf{Contact.Recv}(\mathsf{st}_{\mathsf{bob}}, \mathsf{msg}_{\mathsf{alice}})$

$\mathsf{msg}_{\mathsf{bob}} \leftarrow \mathsf{Contact.Send}(\mathsf{st}_{\mathsf{bob}})$

$\mathsf{If}\ (b = 1) : \mathsf{st}_{\mathsf{alice}} \leftarrow \mathsf{Contact.Recv}(\mathsf{st}_{\mathsf{alice}}, \mathsf{msg}_{\mathsf{bob}})$

$b^* \leftarrow \mathcal{A}^{\mathsf{OUser},\mathsf{ONotify}}()$

$\mathsf{Return} b^*$

In the above game the adversary has access to the following oracles: $\mathsf{OUser}, \mathsf{ONotify}$.

- When called with a single argument, $\mathsf{ONotify}(u)$ runs $\mathsf{msg} \leftarrow \mathsf{Notify.User}(\mathsf{st}_u); \tau \leftarrow \mathsf{Notify.Server}(sk, \mathsf{msg}\tau)$ and returns (msg, τ).
- The OUser Oracle, operates as in section 21.2.2, without ever checking inclusion in $\mathcal{U}$.

Definition 21.2.3. A CCT scheme $(\mathsf{Contact}, \mathsf{Notify}, \mathsf{Scan})$ is private if for every stateful PPT adversary $\mathcal{A}$:

$$\left|\Pr[\mathsf{CCT}_{\mathsf{INT}}^{0}(\mathcal{A}, 1^\lambda)] - \Pr[\mathsf{CCT}_{\mathsf{INT}}^{1}(\mathcal{A}, 1^\lambda)]\right| \approx 0$$

21.3 A Simple CCT protocol

We will describe a simple CCT protocol offering a baseline level of security based on two well-known primitives:

- A digital signature scheme $(\mathsf{GGen}, \mathsf{Sign}, \mathsf{Verify})$ which accepts messages in $\{0,1\}^*$.
- A Pseudorandom function $(\mathsf{KeyGen}, \mathsf{Eval})$.

Definition 21.3.1. A Pseudorandom function (PRF) is a tuple of algorithms $(\mathsf{KeyGen}, \mathsf{Eval})$ with the following syntax:

- A key generator $\mathsf{KeyGen} : \{1\}^* \to \mathcal{K}$.

- An evaluation function $\mathsf{Eval} : \mathcal{K} \times \mathcal{M}_\rho \to \mathcal{P}$.

Definition 21.3.2. A PRF is secure if no PPT adversary $\mathcal{A}$ can distinguish Eval_k (i.e. Eval with the first argument fixed to k) from a random oracle. Concretely, for any PPT $\mathcal{A}$:

$$\Pr\left[k \leftarrow \mathsf{KeyGen}(1^\lambda) : \mathcal{A}^{\mathsf{Eval}_k}(1^\lambda) = 1\right] - \Pr\left[\mathcal{A}^{RO}(1^\lambda) = 1\right] \approx 0$$

For simplicity we will assume $\mathcal{K}, \mathcal{P}$ to be $\{0,1\}^\lambda$ where λ is the security parameter and $\mathcal{M}_\rho$ to be $\{0,1\}^*$.

21.3.1 Contact

First, we describe the layout of a users state st. The state is a tuple $\mathsf{st} = (k, i, M)$, where k is a PRF key, i is a counter (i.e a positive integer), and M is a list of user messages m. User messages m, are tuples $m = (i, \rho)$ of a counter i and a bitstring ρ.

We now describe the opereration of the three interfaces:

- Initialization: $\mathsf{Contact.Init}(1^\lambda) \to \mathsf{st}_0$: Initialises a new state st_0 by calling $k \leftarrow \mathsf{KeyGen}1^\lambda$ and setting $\mathsf{st}_0 := (k, 1, [])$.
- Send: $\mathsf{Contact.Send}(\mathsf{st}) \to (\mathsf{st}', \mathsf{msg}')$ When run with $\mathsf{st} = (k, i, M)$ $\mathsf{Contact.Send}$ computes $\mathsf{msg}' \leftarrow (\mathsf{Eval}(k, i), i)$, calculates its new state $\mathsf{st}' \leftarrow (k, i+1, M)$; it returns $(\mathsf{st}', \mathsf{msg}')$.
- Receive: $\mathsf{Contact.Send}(\mathsf{st}, \mathsf{msg}) \to \mathsf{st}'$ Upon receiving a message msg from another user, it is used to update the state $\mathsf{st} = (k, i, M)$ by appending msg to M i.e $M' \leftarrow M \cup [\mathsf{msg}]$ and returning the new state $\mathsf{st}' \leftarrow (k, i, M')$.

21.3.2 Notify

Notify is the protocol used to inform the server that a user has been diagnosed as infected. The server's *reported state* τ is a tuple (D, σ) where D is list of PRF keys, and σ is a digital signature.

We describe the three interfaces:

- Server Initialization $\mathsf{Notify.Server}(1^\lambda) \to (vk, sk, \tau_0)$. The Server runs $(vk, sk) \leftarrow \mathsf{GGen}(1^\lambda)$, obtaining a public key vk and corresponding secret key sk. The initial server state is $\tau_0 := ([], \epsilon)$. The key vk is distributed to the users.
- User notifying server $\mathsf{Notify.User}(\mathsf{st}) \to \mathsf{msg}$. Given a user state $\mathsf{st} = (k, i, M)$, it returns the message $\mathsf{msg} = k$.
- Server message handling $\mathsf{Notify.Server}(sk, \mathsf{msg}, \tau) \to \tau'$. Server updates $\tau \leftarrow (D, \sigma)$ by appending the message containing the user key, i.e. $D' \leftarrow D \cup [msg]$, producing a new signature $\sigma' \leftarrow \mathsf{Sign}(sk, D')$ and setting $\tau' = (D', \sigma')$.

21.3.3 Scan

Scan is a protocol that is run after contacting the server to obtain a reply τ.

- $\mathsf{Scan}(vk, \mathsf{st}, \tau)$: Parse τ as D, σ. If $\mathsf{Verify}(vk, D, \sigma) \neq 1$, abort. Otherwise, parse D as $k_1, k_2, \ldots, k_n$ and M as $(\rho_1, i_1), (\rho_2, i_2) \ldots (\rho_m, i_m)$. Then calculate all possible $\rho_{j,l} \leftarrow \mathsf{Eval}(k_j, i_l)$ for $j = 1 \ldots n$ and $l = 1 \ldots m$. If, for any j, $\rho_{j,l} = \rho_l$ then return 1, otherwise return 0.

21.4 Security

21.4.1 Correctness

Theorem 21.4.1. *The CCT scheme of section 21.3 is correct given any PRF* $(\mathsf{KeyGen}, \mathsf{Eval})$*, and any correct signature scheme* $(\mathit{GGen}, \mathsf{Sign}, \mathsf{Verify})$*.*

Proof. We note that when Bob contacts Alice, his state is $\mathsf{st}_{\mathsf{bob}} = (k_{\mathsf{bob}}, i_{\mathsf{bob}}, M_{\mathsf{bob}})$. This results in $\mathsf{msg}_{\mathsf{bob}} = (\mathsf{Eval}(k_{\mathsf{bob}}, i_{\mathsf{bob}}), i_{\mathsf{bob}})$ being appended in Alice's list M_{alice}. We also note that no operation[20] can remove items from M_{alice} or alter the value of k_{bob}. This implies that: at the time of Bob running $\mathsf{Notify.User}$, k_{bob} is appended to D, from which no operation will remove it.

Thus, when Alice runs Scan, she obtains a signed list D that contains k_{bob}. By the correctness of the signature scheme, the signature σ will verify. Because k_{bob} is in D, there exists l such that $k_l = k_{\mathsf{bob}}$. Because $\mathsf{msg}_{\mathsf{bob}}$ is in M, there exists j such that $i_j = i_{\mathsf{bob}}$. Thus, there exists an l, j pair such that $\rho_{l,j}$ is contained in M. ∎

We will prove 1-day security for a non-predicting adversary. A non-predicting adversary is one without access to soon-to-be-diagnosed users.

21.4.2 Integrity

We will show that a PPT adversary with oracle access to the PRF (with no access to $\mathsf{Notify.User}$) cannot force Alice's Scan to return 1. We will describe a reduction to the unforgeability of our signature scheme. The main intuition is as follows: if the adversary passes the server's scan message unaltered the probability of Alice outputting 1 is negligible . If the adversary alters the message, he contradicts existential unforgeability. We begin with the first claim.

Theorem 21.4.2. *If,* $(\mathit{GGen}, \mathsf{Sign}, \mathsf{Verify})$ *is existentially unforgerable under chosen message attack, the CCT scheme of section 21.3 has 1-day integrity against non-predicting adversaries with oracle access to the PRF.*

Proof. We start by assuming the adversary chooses to tamper with the server reply, i.e. τ' is such that $D' \neq D$, where $\tau = (D, \sigma)$ was the original server reply.

We build a reduction that plays the role of the adversary in the unforgeability game and the challenger in the integrity game. The reduction starts the unforgeability game by obtaining a verification key vk that will be passed to $\mathcal{A}$ as the server verification key. The reduction is able to play the parts of all users in the system, with the exception of creating τ by signing D. Fortunately, it can use its signing oracle from the unforgreability game. When $\mathcal{A}$ returns $\tau' \neq \tau$, the reduction checks if the signature is valid and if it is, offers it as a forgery in the unforgeability game. If not, it aborts.

The reduction clearly runs in polynomial time. We now evaluate the probability of it succeeding in the unforgeability game. We claim that the probability of wining the unforgeability game is at least equal to that of A wining the CCT integrity game. If $\mathcal{A}$ wins, that implies that the signature on τ' verified, i.e has produced a new signature that verifies under the challenge key vk.

To complete the proof we also need to handle the case where the adversary's choice of D' does not contradict unforgeability, i.e $D = D'$, where $\tau = (D, \sigma)$ was the original server reply.

Claim 21.4.1. *If the adversary sets* τ *such that* $D' = D$*, then the probability that* $\mathsf{Scan}(vk, st_{\mathsf{alice}}{}', \tau') = 1$ *is neglibible.*

Proof. We first note that the list D inside τ contains exactly k_{ian}, which is unknown to the Adversary[21]. Thus, the goal of the adversary is to send Alice a message ρ, i such that $\rho = \mathsf{Eval}(k_{\mathsf{ian}}, i)$. We note that the adversary's view is independent of k_{ian}. We assume that the Adversary made q_e queries to Eval, and (pessimistically) assume he wins if he queries on k_{ian}. This event only happens with probability $q \cdot 2^{-\lambda}$, and as q_e is polynomial in λ is negligible.

[20] The proof also holds if we allow an adversary with an OUser oracle in the experiment.

[21] Technically the Adversary learns it, but can no longer message users.

What remains is to check the probability that Alice's evaluations will match an adversarial message, on the condition that the adversary never queried k_{ian}. We assume the adversary sent q_m messages. Alice will perform at most q_m evaluations, obtaining at most q_m random values, each to be checked against the adversary's q_m messages. The probability of a match is bounded by $q_m^2 \cdot 2^{-}\lambda$. Letting $q = q_m + q_e + 1$ we have that the adversary's probability of success is $q \cdot 2^{-\lambda}$ which is negligible. ■

The claim completes the proof. ■

Stronger Adversaries At this point, we note that without any modifications we cannot prove security for multiple days, or against predicting adversaries. Multiple days allow the adversary to obtain a known-infected key from the server and use that to craft malicious messages. In practice this can be mitigated by making the current time available to the protocol. Predicting adversaries are able to contact Ian (who will end up being diagnosed) and replay his messages towards Alice. This can be (partly) mitigated by challenge-response protocols (with appropriate changes to the syntax). Relaying attacks are harder to mitigate, but may be mitigated by introducing a location parameter to the protocol.

21.4.3 Privacy

Theorem 21.4.3. *The CCT scheme of section 21.3 is private against any*[22] *adversary $\mathcal{A}$ playing the role of the server.*

Proof. We point out that the view of the adversary is not dependent on the value of b. The only thing that depends on the value of b in the experiment are the contents of the M lists inside $\mathsf{st}_{\mathsf{bob}}$ and $\mathsf{st}_{\mathsf{alice}}$. However, that part of the state is never read within the protocol, but only appended to. Since the view of the adversary is independent of b, its output will be independent as well. This implies that $\Pr[\mathsf{CCT}^0_{\mathsf{INT}}(\mathcal{A}, 1^\lambda)] = \Pr[\mathsf{CCT}^1_{\mathsf{INT}}(\mathcal{A}, 1^\lambda)$, thus the scheme is private. ■

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.

[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.

[7] Juan A. Garay and Aggelos Kiayias and Nikos Leonardos, *The Bitcoin Backbone Protocol: Analysis and Applications*, 2015. Advances in Cryptology - EUROCRYPT 2015

[22] The definition requires the adversary to be semihonest i.e honest but curious.