Scribe Notes for CS346

This document is a concatenation of all the scribe notes for the CS346: Cryptography class during the Spring semesters of 2010 and 2012. Some topics might have been skipped in your semester. Therefore, the notes should serve mainly as a supplemental material in studying for the final exam.

1 Motivation

1.1 Why should we learn cryptography?

The following are examples of applications of the topics that we will be studying.

1. Verifying Software

   One common application of cryptography is to verify the author of a software package. For example, when operating system updates are pushed to your computer you want to verify that these updates are in fact from your operating system creators and not from a malicious source. To do this we use a cryptographic primitive called a collision-resistant hash function (CRHF). A collision-resistant hash function is a compressing function for which it is hard to find two arguments that map to the same hash-value. We will examine these functions in more detail later on in the course.

   In the case of software updates, the trusted author will publish the correct hash value of the update and you will compare this value to the hash-value of the update you receive. If we let H be the CRHF, M_C be the correct update, and M_R be the update you received, then we would only accept the update if

   $$H(M_C) = H(M_R).$$

   Since H is collision-resistant, it should be hard for a malicious attacker to find another message that maps to the same hash-value as M_C.

2. Telling Secret Messages

   We will study ways that two parties can communicate publicly without other parties understanding their transmissions.

3. Sharing Research Results

   We will study Zero Knowledge Proofs, which allow you to prove that you have accomplished some goal without revealing any information about how you accomplished it.

1.2 How much is cryptography used in everyday life?

The following are just a few examples of where cryptography appears in our everyday lives.

1. Encrypted BlackBerry Storage
2. SSH - protocol for building secure tunnels to remote consoles
3. SSL/TLS - protocol for building secure tunnels to web servers
4. RFID Security
5. Cellphone Calls
6. Software Updates (Digital Signatures)
7. Quarterback Headsets
8. etc...

2 Background

1. Precision and Rigor

   When making arguments in this class it will be important the we are as rigorous and precise as possible. Often we will begin by expressing the intuition behind an argument, but we must express our intuition with a rigorous mathematical proof using precise cryptographic definitions.

2. Computation Hardness

   In general, we will be interested in cryptographic schemes that can be computed efficiently (in polynomial, $O(n^c)$, time), but take exponential ($O(2^n)$) time to break or find a "collision".

3. Probability

This is a review of some basic probability notation and concepts. For a more complete review of the probability we will be using in this course see Section 1 of `http://www.cs.princeton.edu/courses/archive/spr03/cs511/scribe_notes/0206.pdf`.

Given two events, A and B, we will denote the probability of event A occurring given that B occurs

$$Pr[A|B].$$

Notice, if the two events are independent of one another, then

$$Pr[A|B] = Pr[A].$$

In addition, if the two events are independent, then we can say the following about the probability of both events occurring,

$$Pr[A \cap B] = Pr[A] \cdot Pr[B].$$

We will also use the product rule for probabilities,

$$Pr[A \cap B] = Pr[A]Pr[B|A] = Pr[B]Pr[A|B].$$

Consider the following example. If we know the following probabilities

$$\begin{aligned} Pr[\text{A person is wearing orange}] &= 1/8 \\ Pr[\text{A person goes to UT}] &= 1/4 \\ Pr[\text{A person is wearing orange}|\text{that person goes to UT}] &= 1/2 \end{aligned}$$

then what is the probability that a person is wearing orange and goes to UT?

Using the product rule, we have

$$\begin{aligned} &Pr[\text{A person is wearing orange} \cap \text{goes to UT}] = \\ &Pr[\text{The person goes to UT}]Pr[\text{They are wearing orange}|\text{they go to UT}] \\ &= (1/4)(1/2) = 1/8. \end{aligned}$$

Notice that this is not the same probability that a person is wearing orange given that they go to UT.

3 History of Encryption

Kerckhoff's Principle As stated by Claude Shannon, "The enemy knows the system." In other words, we cannot achieve security through obscurity. We attempt to achieve security while allowing a malicious attacker to know everything that we are doing.

We now explore a few historical encryption systems.

3.1 Shift Cipher

The shift cipher is a primitive encryption scheme used for symmetric communication. In general, encryption schemes for symmetric communication will have the following format.

Phase I Key Setup

The two parties share some secret before starting their communication. We assume that this step is secure, meaning that no attacker will know this secret.

Phase II Encrypt and Share Messages

During this communication phase, an attacker may have access to some or all of the messages being sent.

For the shift cipher, we first choose a secret key, k, in the integers from 0 to 25, representing the 26 letters of the English alphabet. To get the ciphertext, we add the key modulo 26 to each character in the message.

Example: For $k = 3$,

$$PT = YES$$
$$CT = BHV$$

For the first character, 'Y', we have

$$m_1 = 24 + 3 = 27 \equiv 1 \pmod{26} = B$$

A famous example of a shift cipher is Caesar's Cipher, which is a shift cipher with a key of 3. Unfortunately, shift ciphers over the English alphabet are generally insecure because the keyspace is too small. An attacker could easily try all of the 26 possible keys until they uncovered the message.

For example, consider the following message,

$$EMRCVYQ$$

Working forwards and backwards trying to find the key, we can quickly discover that with $k = -2$ the message is

$$GOTEXAS$$

3.2 Alphabetic Substitution

To solve the problem of a small keyspace, we instead consider Alphabetic Substitution. Alphabetic Substitution is simply a permutation of all the letters of the alphabet. In other words, each letter in the alphabet of the message space is mapped to a unique letter in the alphabet. For example, if we were sending genetic sequences with the alphabet AGCT, we could use the following permutation,

$$A = G$$
$$G = T$$
$$C = C$$
$$T = A.$$

Then when we sent a genetic sequence, we would first substitute each letter in the sequence using the rules above to create the ciphertext. Notice that a letter can be mapped to itself, but all of the letters are mapped to a unique letter in the alphabet.

Using this technique with English alphabet gives us 26! possible keys. We have 26 choices to substitute for the letter 'A', then once we have chosen a letter, we have 25 choices to substitute for the letter 'B', then 24 for 'C', and so on. This gives us $26 \times 25 \times \ldots \times 1 = 26!$ or about 10^{26} possible keys.

Unfortunately, this encryption system is still insecure. Even though we do not know the true identity of each character, their frequency in the text remains the same. If someone knew the average frequency of each character in an English text they could use this information to try to determine what each letter in the ciphertext stands for. For example, it is well known that 'E' is the most commonly appearing letter in the English language. Thus, an attacker could find the most commonly appearing letters in a ciphertext and attempt to decode it using 'E' as a substitute. The frequency of some English characters:

$$a \to 8.2\%, \quad e \to 12.7\%, \quad p \to 1.9\%, \quad q \to 0.1\%$$

3.3 Vigenère Cipher

The Vigenère Cipher is just a series of different shift ciphers based on the chosen private key. For example, if we choose the key to be $CAFE$, then we shift the first letter of the plaintext by 2, the second letter by 0, the third by 5, and the fourth by 4 corresponding to the values of the characters in the key. We then repeat these shifts starting with the fifth character of the plaintext, and so on until the entire plaintext is encrypted.

Example:

$$PT = TELLHIMABOUTME$$
$$KEY = CAFE$$

PT =	T	E	L	L	H	I	M	A	B	O	U	T	M	E		
KEY =	C	A	F	E	C	A	F	E	C	A	F	E	C	A	F	E
CT =	W	F	R	Q	K	J	S...									

Notice, that for a key of length t, there are 26^t possible keys that we can choose. Again, we have avoided the small keyspace problem of the usual shift cipher. Since every tth character is shifted by the same amount, if an attacker could figure out the period of the key, t, then attacking the Vigenère cipher becomes equivalent to attacking t separate shift ciphers. Generally t will be short, so the attacker might just try to iterate through the possible values of t beginning with 1.

Alternatively, if the message is long then the attacker can look for repeated phrases in the ciphertext and assume that they correspond to the same plaintext phrase. Using this information the attacker can find a period that would correspond to the separation between different repeated phrases.

4 Symmetric Key Encryption Schemes

Thus far, we have seen specific examples of encryption schemes, but now we will try to define an abstract definition of an encryption scheme and try to precisely define what it means for an encryption scheme to be secure.

We can specify any encryption scheme by 3 randomized algorithms: Gen, Encrypt, and Decrypt. The algorithms are specified as follows:

$$\mathsf{Gen} : n \to K$$
$$\mathsf{Encrypt} : (M, K) \to CT$$
$$\mathsf{Decrypt} : (CT, K) \to M$$

where n is the security parameter, K is the key, M is the plaintext message, and CT is the ciphertext. An encryption scheme is correct only if a ciphertext can be uniquely decrypted to the correct original plaintext. Formally, we say that an encryption scheme is correct if

$$\forall K \in \mathcal{K}, \mathsf{Decrypt}(\mathsf{Encrypt}(M, K), K) = M,$$

where $\mathcal{K}$ is the keyspace. Additionally, we will denote the messagespace, $\mathcal{M}$, and the ciphertextspace, $\mathcal{C}$.

Now, let's consider the notion of secrecy. Intuitively, we want to capture the idea that even if an eavesdropper intercepts our ciphertext and knows how it is made, the ciphertext tells the attacker nothing about the original message. This is called perfect secrecy. Formally, a symmetric key encryption scheme is perfectly secret if

$$\forall c \in \mathcal{C}, \forall m_0, m_1 \in \mathcal{M}, Pr[C = c|m_0] = Pr[C = c|m_1],$$

where c is the encryption of some message in the messagespace. This definition captures the notion that getting a particular ciphertext from encryption is independent of the message. Now we can take a look at an encryption scheme that is perfectly secret.

4.1 One-Time Pad

Let $\mathcal{M}, \mathcal{C}, \mathcal{K} = \{0,1\}^n$ (the n-bit binary strings). The one-time pad is as follows:

1. Gen $\rightarrow K$ chosen uniformly at random from $\{0,1\}^n$
2. $\text{Enc}(M, K) = M \oplus K = CT$
3. $\text{Dec}(CT, K) = CT \oplus K = M$

Correctness is established by verifying that $(M \oplus K) \oplus K = M$. We can establish perfect secrecy by determining the probability of getting a given ciphertext from any message.

Example:

$$\begin{aligned} &\text{Suppose} \\ &CT = 10110 \\ &M = 01101 \\ &\text{then we need} \\ &K = 11011 \end{aligned}$$

Each bit of K has 1/2 probability of being the value we need to get CT from the given M. Since the choice of each bit is independent, for an n-bit string the probability of getting the key we need is exactly $(\frac{1}{2})^n$. So, in general we have for the One-time Pad,

$$\forall c \in \mathcal{C}, \forall m \in \mathcal{M}, Pr[c|m] = \frac{1}{2^n},$$

which shows that the One-Time Pad is perfectly secret.

4.2 Perfect Secrecy

Unfortunately, although the One-Time pad is perfectly secret it is impractical as an encryption scheme. This encryption scheme can only be used once to ensure perfect secrecy, and to do proper encryption the key must be at least as long as the message. Besides the obvious inefficiencies of such a large key size, if two parties can already secretly share a key as long as the message then why wouldn't they just secretly share the message instead of the key!

The failure of the One-Time Pad as a practical encryption scheme, begs the question–can we achieve perfect secrecy where the key is shorter than the message? In other words, if the messagespace is larger than the keyspace can we still achieve perfect secrecy?

Suppose that $|\mathcal{M}| > |\mathcal{K}|$ and that c is a ciphertext that occurs with non-zero probability. In other words, there exists a message, $\tilde{m}$, and a key, $\tilde{k}$, such that $Enc(\tilde{m}, \tilde{k}) = c$ with non-zero probability.

Then, by correctness, there is at most $|\mathcal{K}|$ messages, m, such that $Enc(m, k) = c$ for a given ciphertext $c \in \mathcal{C}$ and any key $k \in \mathcal{K}$. If there were more than $|\mathcal{K}|$ messages that encrypted to c, then there would be at least one key for which more than one message encrypted to c, making unique decryption impossible. Since, $|\mathcal{M}| > |\mathcal{K}|$ there must be at least one message, m', that cannot encrypt to c, meaning $Pr[c|m'] = 0$. Then, the probability of getting c is not independent of the message because $Pr[c|\tilde{m}] \neq Pr[c|m']$. Hence, we cannot achieve perfect secrecy.

5 Background Review

Today, we will discuss how to prove that encryption schemes are secure. In general, we will attempt to prove statements of the following form, "Any efficient attacker/algorithm can break our system with at most negligible probability." First, let's review some definitions and notation.

5.1 "Big Oh" Notation

Suppose we have a function $f(n) : \mathbb{N} \to \mathbb{N}$. Then we say that, $f(n) = O(g(n))$ if and only if there exists constants $d, n_0 \in \mathbb{Z}$ such that,

$$\forall n \geq n_0, f(n) \leq d \cdot g(n).$$

Examples :

$$n^2 + 5n = O(n^3)$$
$$n^5 = O(0.000001n^5), d = 1/0.000001$$

5.2 Polynomials

We say that $f(n) : \mathbb{N} \to \mathbb{N}$ is a **polynomial** if and only if $f(n) = O(n^c)$ for some $c \in \mathbb{Z}$.

Examples :

$$n^6, n^{1000}, \text{etc.}$$

In general, we say that an algorithm is **efficient** if it is polynomial.
A function that is not polynomial is **superpolynomial**.

Examples :

$$n!, 2^n, n^{\lg n}, \text{etc.}$$

We say that a function, $f(n) : \mathbb{N} \to \mathbb{N}$ is **negligible** (negl.) if for all polynomials, p, there exists $n_0 \in \mathbb{Z}$ such that,

$$\forall n \geq n_0, f(n) < \frac{1}{p(n)}.$$

Examples :

$$0, \frac{1}{2^n}, \text{etc.}$$

6 Encryption Security

Now let's explore one definition of security for encryption schemes that we will be using. This definition takes into account a particular type of adversarial attack called a chosen plaintext attack (CPA). CPA security captures the notion of an adversary who has the ability to eavesdrop on arbitrary messages between a sender and receiver before attempting to decrypt a message. Recall that we can define any encryption scheme, Π, by three algorithms, Π = (KeyGen, Enc, Dec).

6.1 Indistinguishability under Chosen Plaintext Attacks (IND-CPA)

First we define a "security game" that is played between the attacker, $\mathcal{A}$, and a challenger.

1. The challenger generates a key, k, by running KeyGen(n), where n is the security parameter.
2. $\mathcal{A}$ is given "Oracle Access" to Enc(k, $\cdot$). This means that $\mathcal{A}$ can get the encryption for a polynomial, Q, number of messages.

3. $\mathcal{A}$ outputs two messages $m_0, m_1 \in \mathcal{M}$.
4. The challenger chooses $b \in \{0,1\}$ uniformly at random and sends the challenge ciphertext, $c^* = \text{Enc}(k, m_b)$, to $\mathcal{A}$.
5. $\mathcal{A}$ is again given "Oracle Access" to $\text{Enc}(k, \cdot)$.
6. Attack outputs a guess, b'.

We define the **advantage** of the algorithm/attacker $\mathcal{A}$ as follows,

$$Adv^{cpa}_{A,\Pi} = Pr[b' = b] - 1/2 = 1/2(Pr[b' = 1|b = 1] - Pr[b' = 1|b = 0]).$$

We say that an encryption system is **IND-CPA secure** if for all efficient algorithms, $\mathcal{A}$, $Adv^{cpa}_{\mathcal{A},\Pi} = negl(n)$, where n is the security parameter.

If an encryption scheme is deterministic (the Enc algorithm is deterministic) then there is a unique, consistent encryption for every message. A deterministic encryption scheme cannot be secure since we can simply ask for the encryption of the two challenge messages during the oracle access step and compare the oracle's response to the challenge ciphertext.

7 Determinism in Encryption

Last time we saw a precise definition of security. We saw a very robust definition that errs on the side of caution with respect to the power of the adversary. You should review and feel comfortable with the definition of IND-CPA security. In real life an attacker may not have the abilities he has in the IND-CPA game, but the attacker might in certain applications and we want to capture these situations. We also discussed why a deterministic encryption scheme cannot be IND-CPA secure. Let's look at a concrete example of a deterministic scheme. Suppose that an adversary was trying to figure out where we are meeting for lunch,

Example :

$$\begin{aligned} \text{yesterday } Enc(\text{“}MeetatWendy's\text{''}) &= C \\ \text{today } Enc(\text{“}MeetatWendy's\text{''}) &= C' \end{aligned}$$

In a deterministic encryption scheme $C = C'$ because for any given input the output is always the same. So if the adversary knows that the we ate at Wendy's yesterday, then the adversary knows that we will be eating there again. However, our definition of IND-CPA security captures this type of attack by giving the attacker access to the encryption oracle. So, no IND-CPA secure scheme can be deterministic.

8 Can we be sure of security?

If an encryption scheme is secure, then why do we say that there is a negligible chance of breaking it and why do we restrict ourselves to only polynomial-time attackers? We allow a negligible chance of breaking our scheme since the adversary can always, with negligible probability, *guess* the key we are using and easily decrypt any message. In addition, we require attackers to be efficient to eliminate the possibility of *brute-force* attacks. A brute-force attack is when the attacker simply tries every value in the keyspace until he thinks he has correctly decrypted the messages. Next, we will explore what it means for an attacker to be "efficient" and whether this is a reasonable restriction.

8.1 P = NP?

Let's review some computational complexity theory that will give us insight into some of our security assumptions. We informally review the following two complexity classes,

P = Languages that can be solved in polynomial time. i.e., decision problems with polynomial time solutions. For example, graph connectivity.

NP (non-deterministic polynomial time) = Problems that, given a "hint" (witness), can be solved in polynomial time. For example, three color.

There are no known polynomial time solutions to any problems in **NP**. However, there are exponential time algorithms for solving these problems. For example, consider the graph three color problem. We believe that it is hard to find a coloring of the graph, but that given a coloring it is easy to check in polynomial time.

If $\mathbf{P} = \mathbf{NP}$ then any problem that we can verify easily, we can solve easily. In cryptography, the key of an encryption scheme can be verified easily, but it should take exponential time to find the key.

Given key, k, and

$$c_1 = Enc(k, m_1), c_2 = Enc(k, m_2), \ldots, c_{100} = Enc(k, m_{100})$$

we can check if k is correct by decrypting each message. However, while it is easy to verify k, the security of our systems rely on the belief that it is hard to find k, or that $\mathbf{P} \neq \mathbf{NP}$. In cryptography, all of our systems will be based on this assumption.

We could just assume that all of our systems are secure, since we cannot prove security absolutely. However, we will try to establish the security of small constructions based on the weakest assumptions possible and then build on these constructions to build big cryptographic schemes. So, we want to build a big system off the security of some small building blocks. We will typically begin with an assumption and then build a secure primitive and then we will use the security of this primitive to prove the security of the larger system.

We will start by building an encryption scheme from a primitive called a pseudo-random function (PRF).

9 Pseudo-Random Function (PRF)

Before we define a pseudo-random function we must know the definition of a random function. We can think of a random function $f : \{0,1\}^k \rightarrow \{0,1\}^l$ as table with inputs in $\{0,1\}^k$ and randomly chosen outputs in $\{0,1\}^l$. This means that a random function is deterministic. For any input value in the table we will always get the same output. So we are randomly choosing a function from k bits to l bits, but the function itself is deterministic.

To describe a random function we need a table of size

$$2^k l \text{ bits},$$

where 2^k is the number of input rows we need and l is the length of each output value we have stored at that position in the table. So, we can think of a random function as a table of 2^k l-bit numbers.

Now, let's define a pseudo-random function. For a PRF we choose a random key, K, for the function

$$F_K : \{0,1\}^k \rightarrow \{0,1\}^l, \text{where } |K| << 2^k l.$$

We want this function to look like we are choosing a huge random table of size $2^k l$ bits, but we actually just want to choose a small key of size $|K|$ and then evaluate the PRF, F_K.

So, for security we will require that we cannot tell the difference between a truly random function, f, and a PRF, F_K. Formally,

$$\forall \text{polytime algorithms } \mathcal{A}, Pr[\mathcal{A}^{f(\cdot)} \text{ outputs 1}] - Pr[\mathcal{A}^{F_K(\cdot)} \text{ outputs 1}] = negl(n),$$

where n is the security parameter. Informally, when $\mathcal{A}$ is given oracle access to each function, we want the difference in $\mathcal{A}$'s output between the two functions to be negligible.

Why is the PRF an interesting primitive? We will see that we can use a PRF to build a simple encryption scheme. This will be our first construction of an encryption scheme and our first security proof. We will build our scheme off of PRFs and show that if our PRF is secure then our scheme is secure. This technique is called a reduction.

10 Encryption from PRFs

Let F_K be a PRF, $F_K : \{0,1\}^k \rightarrow \{0,1\}^l$. We define our encryption scheme as follows,

- KeyGen(n):
  1. Choose a random PRF key, $K \in \{0,1\}^n$
  2. Set K as the secret key of the encryption scheme.
- Enc(K,$M \in \{0,1\}^l$):
  1. Choose $r \in \{0,1\}^k$ uniformly at random.
  2. $C = (C_1 = r, C_2 = M \oplus F_K(r))$
- Dec(K, $C = (C_1, C_2)$):
  1. $M = C_2 \oplus F_K(C_1)$

where n is the security parameter. We call $F_K(r)$ a **blinding factor**. Use XOR to "blind" the message, or to make it look random.

Intuitively, why is this encryption scheme secure? If $F_K(\cdot)$ was a truly random function, this would be a One-Time Pad. So, assume the attacker cannot tell the difference between a PRF and a truly random function, then this scheme must be secure. In addition, we need to make sure that we do not use the same random input to the PRF, or else we could use this to determine information about two different messages. For two particular ciphertexts, $C = (C_1, C_2)$ and $C' = (C'_1, C'_2)$, the probability that we use the same random input is,

$$Pr[C_1 = C_1'] = \frac{1}{2^k}.$$

Hence, we want k to be big so that we don't choose the same r twice. Next time, we will formally prove security for this encryption system.

11 Random Table Encryption

In this lecture we will give a proof of security of our PRF encryption scheme using a reduction.

Before we begin our proof, we will construct a contrived encryption scheme that will help us to prove the security of our system. This scheme, which we will call Random Table Encryption, will capture the similarities of our scheme to a One-Time pad. We define the system as follows,

- KeyGen(n):
  1. Choose a random table (function) $f(\cdot)$ of $2^k l$ bits.
- Enc(K,$M \in \{0,1\}^l$):
  1. Choose $r \in \{0,1\}^k$ uniformly at random.
  2. $C = (C_1 = r, C_2 = M \oplus f(r))$
- Dec(K, $C = (C_1, C_2)$):
  1. $M = C_2 \oplus f(C_1)$.

Recall that a PRF is secure if and only if the following holds for any polynomial time attacker, $\mathcal{B}$,

$$|Pr[\mathcal{B}^{F_K(\cdot)} \rightarrow 1] - Pr[\mathcal{B}^{f(\cdot)} \rightarrow 1]| = negl(n).$$

We will use this definition of a PRF to show that our original system is indistinguishable from the Random Table Encryption scheme and then prove that the Random Table Encryption scheme is information theoretically secure.

12 Security Proof

Many security proofs that we will do in this class will follow the same general outline as the proof of security for our PRF encryption scheme. Here is the general outline we will use for a proof of security for an encryption scheme.

12.1 Outline of the Proof

1. Define an alternate encryption scheme. (e. g., Random Table Encryption)
2. Show alternate system can only be negligibly more secure than the original system.
3. Show that any (unbounded) attacker has at most negligible success in breaking the alternate system.

We will begin the proof with a Lemma that shows the Random Table Encryption system can only be negligibly more secure than the real encryption system.

Lemma 12.1. *For any polynomial time algorithm, $\mathcal{A}$, if*

$$Pr[Success^{IND-CPA}_{A_{real}}] - Pr[Success^{IND-CPA}_{A_{randomtable}}] = \varepsilon$$

then there exists an algorithm $\mathcal{B}$ such that

$$Pr[\mathcal{B} \text{ outputs 1} \mid \text{oracle to PRF }] - Pr[\mathcal{B} \text{ outputs 1} \mid \text{oracle to random table }] = \varepsilon.$$

Proof. Given an attacker $\mathcal{A}$ that has the above property, see Algorithm 1 for the construction of $\mathcal{B}$ that attacks the PRF game.

Claim 12.2.

$$Pr[\mathcal{B} \rightarrow 1 | \mathcal{O} \text{ to PRF }] = Pr[\mathcal{A} \text{ success in Real System }]$$
$$Pr[\mathcal{B} \rightarrow 1 | \mathcal{O} \text{ to Random Table }] = Pr[\mathcal{A} \text{ success in Random Table System }]$$

This claim follows directly from the construction of $\mathcal{B}$. If we subtract the two parts of the claim from each other we get the Lemma. □

Algorithm 1 $\mathcal{B}$

1: $\mathcal{B}$ has oracle access to $\mathcal{O}$ – either a PRF or a Random Table/Function
2: $\mathcal{B}$ "starts" $\mathcal{A}$:
3: $\mathcal{A}$ asks for the encryption of messages $m_1, \ldots, m_Q$
4: For each m_i, $\mathcal{B}$ chooses $r_i \in \{0,1\}^k$ and sets $c_i = (r_i, \mathcal{O}(r_i) \oplus m_i)$
5: $\mathcal{A}$ gives m_0, m_1
6: $\mathcal{B}$ picks a random bit $b \in \{0,1\}$, and $r^* \in \{0,1\}^k$ and sends the challenge ciphertext $c^* = (r^*, \mathcal{O}(r^*) \oplus m_b)$ to $\mathcal{A}$.
7: Repeat the query phase from step 3
8: Get guess b' from $\mathcal{A}$
9: $\mathcal{B}$ outputs "1" ("It's a PRF") if $b = b'$ ($\mathcal{A}$ was right)

Now that we have shown that the two systems are indistinguishable, we still need to show that the Random Table game is secure. In other words, what is

$$Pr[\mathcal{A} \text{ succeeds in Random Table System }]?$$

There are two interactive phases in the Random Table game that we must consider, the Query Phase and the Challenge Phase. Recall,

Query Phase: $r_1, \ldots, r_Q$, where $c_i = (r_i, f(r_i) \oplus m_i)$

Challenge Phase: $c^* = (r^*, f(r^*) \oplus m_b)$

For all i, we don't want $r^* = r_i$, or else $f(r^*)$ will be known by the attacker and so the system will be insecure. Now, let's consider the two possibilities for the randomly selected r^*,

Case I:(Good) $\forall i, r_i \neq r^*$

Case II:(Bad) $\exists i, r_i = r^*$

Since, one of these cases must occur, we can rewrite the probability of success in the Random Table game as follows,

$$\begin{aligned} Pr[\text{success of } \mathcal{A}_{RT}] &= Pr[\text{success} \mid \text{Case I }]Pr[\text{Case I }] + Pr[\text{success} \mid \text{Case II }]Pr[\text{Case II }] \\ &\leq \frac{1}{2} \cdot 1 \text{ (in the worst case)} + 1 \cdot \frac{Q}{2^k} \text{ (by the Union Bound)} \\ &= \frac{1}{2} + \frac{Q}{2^k} = \frac{1}{2} + negl(n) \end{aligned}$$

Recall from Lemma 12.1, that if our PRF is secure then there is a negligible difference, ε, between any attackers success in the Real game and their success in the Random Table Game. Thus, if we combine this result with the result from Lemma 12.1, we get that,

$$\begin{aligned} Pr[\text{ success in the Real game }] &= Pr[\text{ success in the Random Table game }] + \varepsilon \\ &\leq \frac{1}{2} + \overbrace{\frac{Q}{2^k} + \varepsilon}^{\text{negligible}} = \frac{1}{2} + negl(n). \end{aligned}$$

So, the Real encryption system is secure.

13 Review

First let's review the cryptographic structures that we have seen thus far. So far, we have only seen one provably secure encryption scheme, that we constructed using PRFs. Recall that we reduced our encryption scheme to a similar scheme using a truly random function instead of a PRF. Then we showed that the game with the truly random function was information theoretically secure. In other words, we reduced the security of our encryption scheme to the security of the PRF. This is captured in the following theorem, that follows directly from our proof of security.

Theorem 13.1. *Suppose that there exists a polynomial time attacker, $\mathcal{A}$, that has a non-negligible advantage in breaking our IND-CPA system then there exists a polynomial time attacker, $\mathcal{B}$, that has a non-negligible advantage in distinguishing a PRF from a truly random function.*

For our encryption scheme we chose to use PRFs as our building block. In general, we can choose our building blocks from two places,

1. Number Theory
2. Heuristics

In this lecture we will focus on a heuristic approach called Block Ciphers.

14 Block Ciphers

Block Ciphers are sometimes referred to as Pseudorandom Permutations (PRPs), which gives us some intuition about how they function. Recall how a PRF functions,

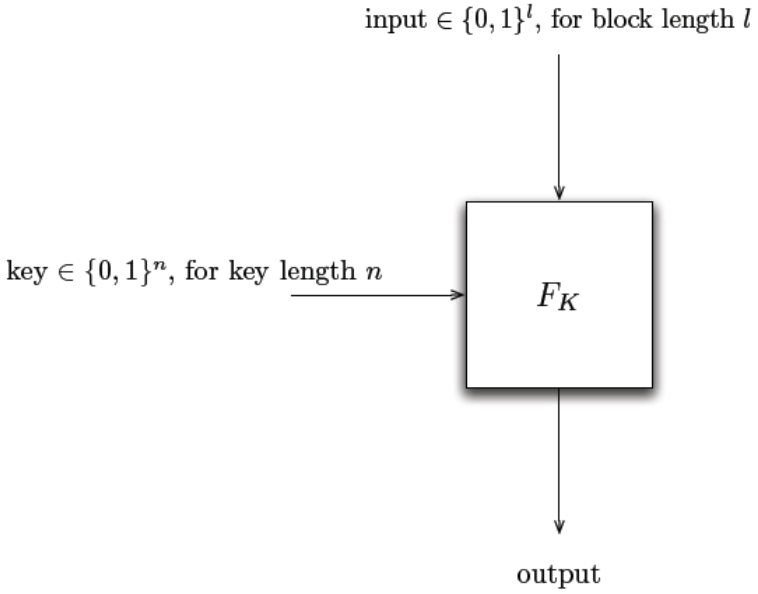

Figure 1: Functionality of a PRF.

A PRP functions similarly to a PRF with the additional restriction that the function, F_K, must be a permutation. The properties of a permutation are,

1. One-to-One
2. Onto
3. Domain = Range

We can think of a permutation as a "reordering" of the elements in the domain. So, the name Psuedo-random Permutation tells us that Block Ciphers have the following properties,

1. Permutation
2. Indistinguishable from a truly random permutation

Just like PRFs, using PRPs gives us a huge advantage in the number of bits needed over a truly random permutation. However, for l-bit inputs, there are 2^l different inputs, so there are $2^l!$ different permutations, but there are only $2^K!$ different PRPs.

Recall that our definition of security for a PRF was that it was indistinguishable from a truly random function, formally

$$\forall \text{ polynomial time } \mathcal{A}, Pr[\mathcal{A}^{F_K} = 1] - Pr[\mathcal{A}^f = 1] = negl(n).$$

Since PRPs are a specific type of PRFs, we will use this same definition of security with the additional constraint that F_K and f be permutations.

Over the course of the next two lectures we are going to look at two different Block Cipher designs. Today we are going to look at the Substitution-Permutation design.

15 Substitution-Permutation

Substitution-Permutation Block Ciphers consist of 3 separate phases,

1. Key Mixing Phase, XOR round input with (part of) key
2. Substitution Boxes (S-boxes)
3. Mixing or Permutation.

Figure 2 shows the general functionality of a Substitution-Permutation Block Cipher. The S-box permutations will be known to the attacker and in general, the cipher will continue for approximately 10 rounds.

We will want two different principles in our Block Cipher scheme:

1. Invertibility of S-boxes
2. "Avalanche Effect"/"Diffusion Principle"
   - Want changes in input to effect entire permutation:
   - S-box: changing 1 bit changes ≥ 2 bits of output
   - All output bits of an S-box go to a different S-box in next round
   - Enough Rounds

Remember, these are heuristics and NOT proofs of security. These are intuitive guesses about what makes a secure system.

What happens if we only have 1 round, as seen in Figure 3?

In this scheme we can go backwards to get y_2, using the inverse permutation. Then, we can invert each S-box to get $y_1 = k_1 \oplus x$. Finally, take $y_1 \oplus x = k_1$ to get the key! For another example of too few rounds, see the 2 round attack in Katz and Lindell.

16 Advanced Encryption Standard (AES)

- DES was 56-bit keys (too small!)
- 1997: NIST held a competition for new Block Ciphers
  - 30 entries
  - 5 finalists (they thought all were secure)
  - Allowed entrants to attack each other's systems

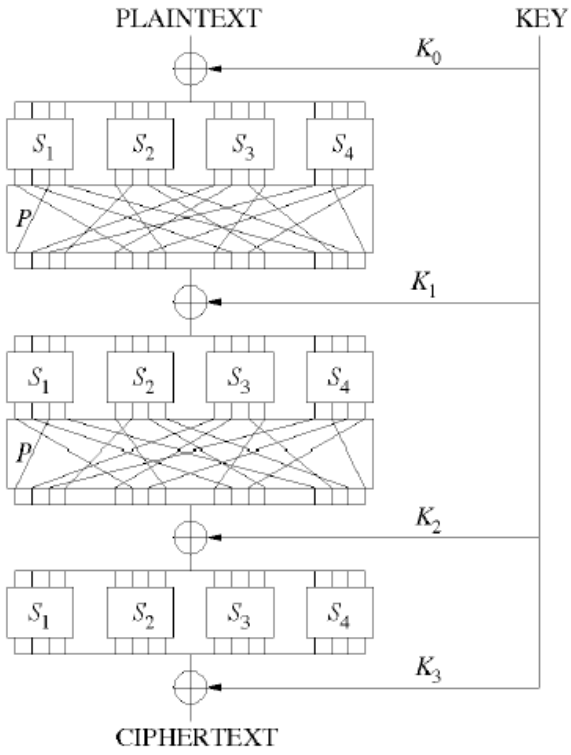

Figure 2: A sketch of a Substitution-Permutation Network with 3 rounds, encrypting a plaintext of 16 bits into a ciphertext of 16 bits, using a block size of 4. The S-boxes are the S_is, the P-boxes are the same P, and the round keys are the K_is. From Wikimedia Commons.

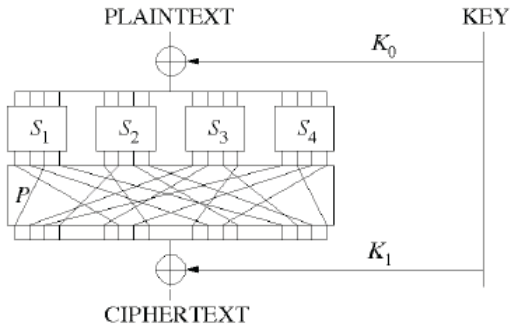

Figure 3: One round of a Substitution Permutation Block Cipher. Edited original from Wikimedia Commons.

- 1999: Selected Rijandel Cipher as the winner and based AES on this.
- Substitution-Permutation type cipher
  - 128 bit fixed block size
  - 128 bit key - 10 rounds
  - 192 bit key - 12 rounds
  - 256 bit key - 14 rounds

Normally we discuss schemes with an arbitrary security parameter that decides the parameter sizes. In practice specific parameter sizes are fixed.

AES organizes input into a state made up of a 4×4 matrix of blocks of bits. The same S-box is used on each block (allowing for hardware optimization), as seen in Figure 4, and rotation (using row and column shifts) is done during the permutation phase. Finally, there is a key scheduler that is used to determine the key at the beginning of each round.

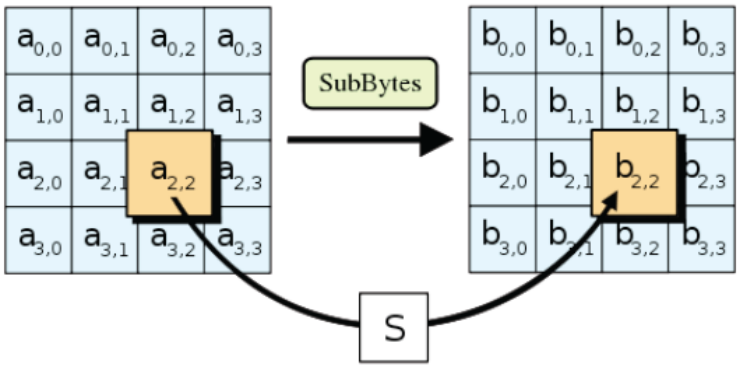

Figure 4: Each byte in the state is replaced with its entry in a fixed 8-bit lookup table, S, so that $b_{i,j} = S(a_{i,j})$. From Wikimedia Commons.

17 Feistel Networks

We want to build a permutation network from non-invertible s-boxes.

1. Break input into 2 halves L_0, R_0
2. For each round: $L_i = R_{i-1}, R_i = L_{i-1} \oplus f_{k_i}(R_{i-1})$
3. f_{k_i} is a "mangler function" that depends on the key in each round.

See Figure 5 for an example of Feistel Network.

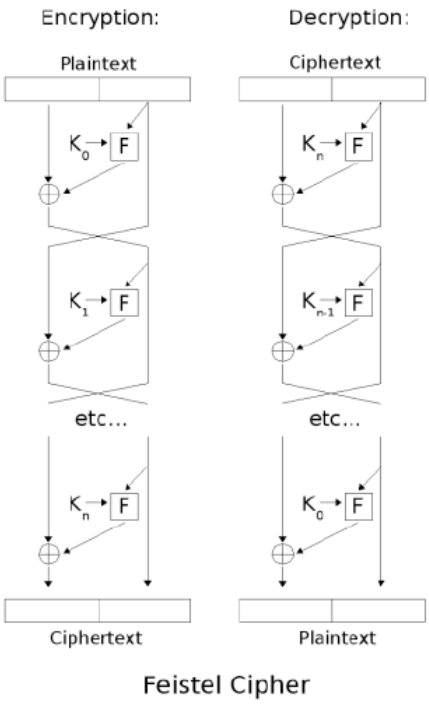

Figure 5: Encryption and Decryption for a Feistel Network.

Inverting

To invert the permutation, we have

$$R_{i-1} = L_i \quad L_{i-1} = R_i \oplus f_{k_i}(R_{i-1}),$$

as depicted in Figure 5.

The s-boxes (f_{k_i}) are not necessarily invertible. Unlike AES, we don't need to invert the s-box to get back to the input.

Let's take a look at a well-known Encryption Scheme that employs a Feistel Network.

18 DES - Data Encryption Scheme

- Developed in 1970's by IBM and the NSA.
- 1977 became FIPS Standard
- 56 bit keys, 64 bit block size, 16 rounds
- Brute Force Attacks can break DES now, but it withstood many challenges.

"Mangler Function"

1. Each round "derive" 48-bit round key from 56-bit key
2. Expand 32-bit input to 48-bit input
3. XOR in key
4. 8 s-boxes from 6 to 4 bits giving 32-bit output
5. Permute the output

See Figure 6 for a visual overview of the DES mangler function. Again, note that the s-boxes, S_i, are NOT invertible. DES was given to the NSA by IBM, and the NSA gave it back with all of the s-boxes changed. This gave rise to suspicion that there was a back door built into the s-boxes by the NSA.

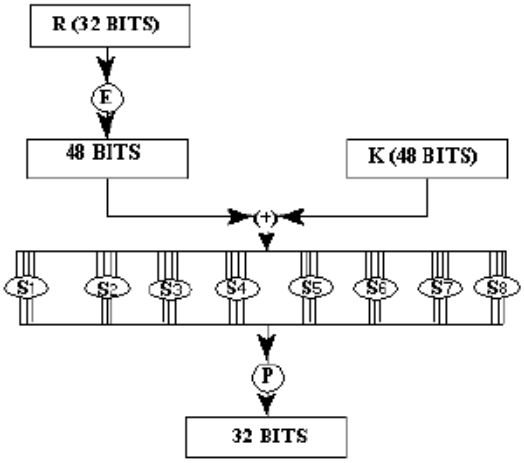

Figure 6: The DES Mangler Function. Image taken from the original Federal Information Processing Standards(FIPS) Publication of DES.

Attempts to Attack DES

In the 1980's Biham and Shamir developed Differential Cryptanalysis. They showed that differences in output lead to differences in input between two different input output pairs, $(x_1, y_1 = F_K(x_1))$ and $(x_2, y_2 = F_K(x_2))$, with probability $1/2^l$. However, it takes approximately 2^{40} plaintext pairs to find a break in the mangler function.

Most cryptographers believe that the NSA probably knew about differential cryptanalysis but did not want to let anyone know about it. It turns out that if the s-boxes were slightly different then they would have been susceptible to this type of attack.

Despite the fact that DES has withstood many attacks, modern processing power makes the a brute force attack possible on the 56-bit key size. DES can now be broken in a few minutes. Now we will examine some ways to make it more secure, in order to extend its life.

Increase the Key Length (Double Encryption)

We now let the mangler function be $F'_{k_1k_2}(x) = F_{k_2}(F_{k_1}(x))$, where $F'_{k_1k_2}$ is a pseudorandom permutation or block cipher (NOT an Encryption Scheme).

Since $k = k_1k_2$, we have a 112-bit key, so we would hope for 112-bit security. Is there an attack that takes $<< 2^{112}$ time?

"Meet in the Middle Attack"

Suppose that the key length is the same as the block length, n. Suppose that an input and output pair, (x, y) is known, and an attacker wants to find the key $k = k_1 k_2$. Let $F_{k_1}(x) = z = F_{k_2}^{-1}(y)$ be the intermediate value during the double encryption, as shown

$$x \overset{F_{k_1}}{\rightarrow} z \overset{F_{k_2}}{\rightarrow} y.$$

Then, the attack proceeds as follows,

1. Attacker gets 1 input/output pair x, y
2. Tries all keys from left to get set T of intermediate values $(z, guessk_1)$
3. Tries all keys from right to get set T' of intermediate values $(z', guessk_2)$
4. Sort the list; whenever z is both on the left and right, add $k_1, k_2)$ to the set of possible keys, S
5. After 1 try $2^{2n}/2^n = 2^n$ matches. Then if the attacker repeats this process one or two times it will the set of possible keys will be narrowed down completely.

With no knowledge of the key there are 2^{112} possible key pairs, but we group the entries in T and T' by their z values and only keep the guessed key pairs when $z = z'$. Then we need to find the correct pair of keys out of these positive matches. So, we started with 2^{2n} possible keys, where n is the original 56-bit key length. The chances of a false positive match is the same as the probability of finding two different values in either permutation that map to the same value, or $1/2^n$. So there will be 2^n positive matches. This cuts in half the key space and we can continue to use this method to decrease the possible keys.

So, Double Encryption is not good enough. What should we try next?

Triple Encryption

Again, we increase the number of permutations in the mangler function, so $F'_{k_1 k_2 k_3}(x) = F_{k_3}(F_{k_2}(F_{k_1}(x)))$.

$$x \overbrace{\underbrace{\overset{F_{k_1}}{\rightarrow} z_1}_{\text{meet after 1}} \overset{F_{k_2}}{\rightarrow} z_2}^{\text{meet after 2}} \overset{F_{k_3}}{\rightarrow} y$$

Now if we try to use the mapping of x and the inverse of y to "meet in the middle", we must choose to either meet after the first encryption of after the second encryption. But this means, that either from the left hand side or the right hand side we must do 2 encryptions, giving us 2^{112} security. This variant is called Triple DES or 3 DES. We will not try to prove security for this system.

19 Modes of Encryption

Last class we looked at block ciphers/psuedorandom permutations. Remember, block ciphers are not an encryption scheme. Now that we have seen what they are, we want to use them to build an encryption scheme, just like we did using psuedorandom functions.

We will try to prove the security of an encryption system, using the psuedorandom permutation (PRP) as a black box. There are many different types of encryption schemes, called modes, that use PRPs as the foundation of their security.

For all of the following modes, we will use the notation,

- n - keysize
- l - block length
- t - encrypted blocks

In addition, we will assume that messages are multiples of the block length. This assumption is reasonable because we could always just pad the end of the message with some standard character to make it the appropriate length without effecting the security of the scheme.

19.1 Electronic Codebook Mode (ECB)

Suppose we are given a message $M = M_1 M_2 \dots M_t$ and a PRP F_K, then ECB encodes M, as follows,

$$C_i = F_K(M_i),$$

where C_i is the ith block of the ciphertext. Figure 7 gives a graphical depiction of this encryption scheme.

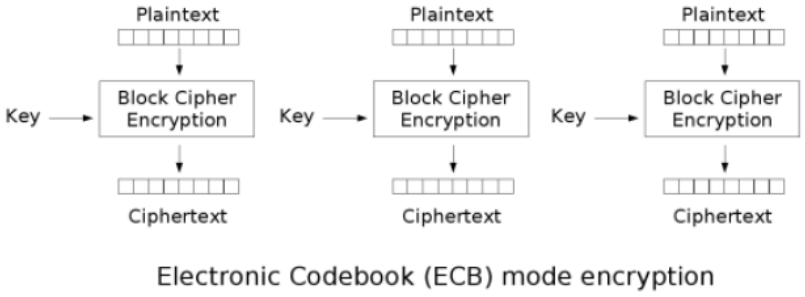

Figure 7: Electronic Codebook Mode encryption. Image taken from Wikimedia Commons.

Is this scheme IND-CPA secure?

19.1.1 Attack

Consider the following attack, let $m_0 = a^l a^l$ and $m_1 = a^l b^l$, then

$$a^l \to F_K \to C_1$$
$$a^l \to F_K \to C_2, C_1 = C_2$$

$$a^l \to F_K \to C_1$$
$$b^l \to F_K \to C_2, C_1 \neq C_2.$$

Encrypting the same plaintext block always results in the same ciphertext. So, by encrypting the same plaintext block twice in a row, we will get two equal ciphertext blocks. Using this fact, we can guess b 100% of the time. We did not use the query phase in this attack.

What if the two challenge plaintexts had to be,

$$m_0 = AB, m_1 = XY?$$

An attacker could ask $m' = AB$ in the query phase and if $E(m') = E(m_b)$ then $b = 0$.

So, ECB also fails because the encryption scheme is deterministic, so AB will always encrypt to the same thing. Using the query phase we can always break schemes that are deterministic.

19.2 Cipher Block Chaining (CBC)

Now we will consider a scheme that is non-deterministic. First, we choose an "Initialization Vector", IV, of the block length, l, uniformly at random from $\{0,1\}^l$. Then, we encrypt the message $M = M_1 \dots M_t$ as follows,

$$\begin{aligned} C_1 &= F_K(M_1 \oplus IV) \\ C_i &= F_K(M_i \oplus C_{i-1}), \text{ for } i > 1 \\ CT &= (IV, C_1, \dots, C_T). \end{aligned}$$

See Figure 8 for a visual representation of the CBC mode.

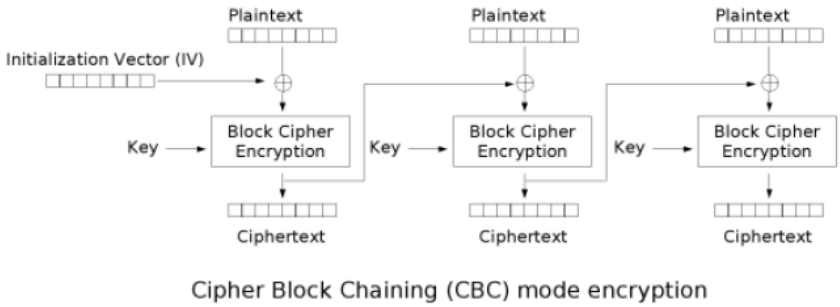

Figure 8: Cipher Block Chaining Mode encryption. Image taken from Wikimedia Commons.

Unlike ECB, it turns out that this scheme is IND-CPA secure. We omit the proof here. This scheme cannot do parallel encryption because each block must wait for the encryption of the previous block. However, we can do random access decryption of the ith block using C_i and C_{i-1}, as follows,

$$M_i = C_{i-1} \oplus F_K(C_i).$$

Suppose that evaluating F_K on a block is expensive, but it is cheap to take $\oplus$ of two blocks (this is generally true). Can we do any work ahead of time to make online encryption cheap? This is called, offline encryption. Offline encryption is done before you see any message queries, whereas online encryption is done after the messages are seen. For CBC, we cannot do any offline encryption since without the message blocks we cannot do any of the $\oplus$ operations.

19.3 Output Feedback Mode (OFB)

For OFB, we again select an initialization vector, $IV \in \{0,1\}^l$. Then, we encrypt as follows,

$$\begin{aligned} S_1 &= F_K(IV) \\ S_i &= F_K(S_{i-1}), i > 1 \\ C_i &= M_i \oplus S_i, i \geq 1 \\ CT &= (IV, C_1, \dots, C_T). \end{aligned}$$

See Figure 9 for a visual representation of OFB mode.

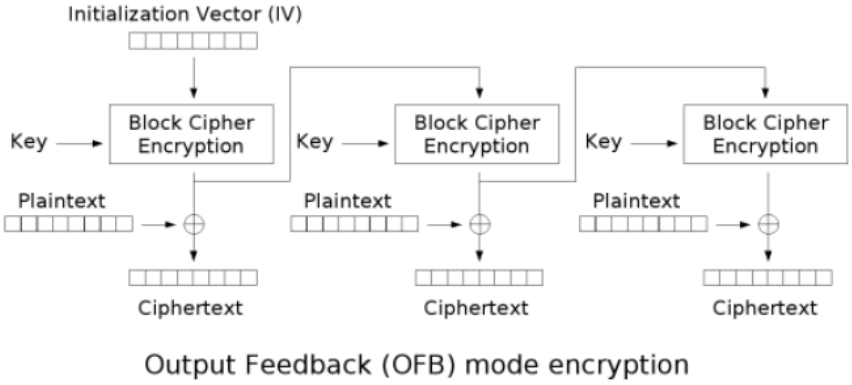

Figure 9: Output Feedback Mode encryption. Image taken from Wikimedia Commons.

The S_i's do not depend on the message, so we can compute these offline and then only compute the $\oplus$ operations for any given message. However, we cannot do parallel encryption or random access decryption because each block is dependent on the last. However, if we do the offline processing first, then we can do the $\oplus$ operations in parallel and do random access decryption using the stored S_i values.

19.4 Counter Mode (CTR)

Again, we choose an $IV \in \{0,1\}^l$. Then we encrypt as follows,

$$C_i = M_i \oplus F_K(IV + (i-1))$$
$$CT = (IV, C_1, \ldots, C_T).$$

See Figure 10 for a graphical depiction of CTR mode.

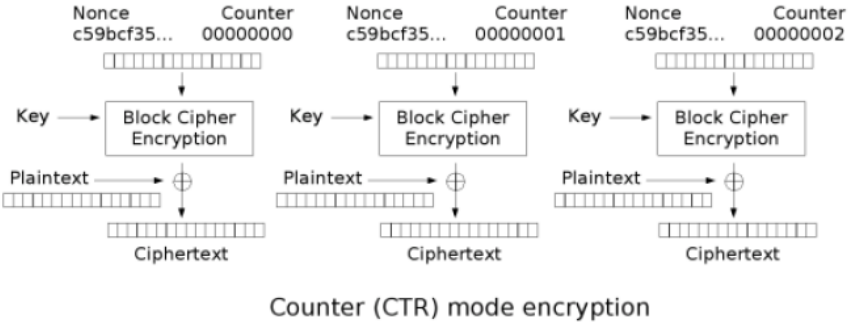

Figure 10: Counter Mode encryption. In this example, *nonce*, is the same as the initialization vector, IV. Image taken from Wikimedia Commons.

20 Summary

Here is a summary of the properties of each encryption mode that we covered:

Mode	IND-CPA	Parallel Encryption	Random Access Decryption	Offline Encryption
ECB	×	-	-	-
CBC	√	×	√	×
OFB	√	×	×	√
CTR	√	√	√	√

21 Security Proof of CTR

Last class we saw block cipher modes, however, we did not give any formal proofs for the security of these encryption schemes. Today, we will prove IND-CPA security for counter (CTR) mode encryption.

We will define an alternate encryption scheme, *system*$'$, that we will use in our security proof. Note that this scheme does not necessarily have to be efficient, since it will never actually be used in practice–the scheme is only used for the sake of the security proof.

CTR Mode with Truly Random Function

- KeyGen: Chooses a random permutation, F. In practice this takes a HUGE number of bits! The table size will be $2^l l$ bits.
- Enc($M = M_1 \dots M_T$):
  1. Choose uniformly at random $IV \in \{0,1\}^l$
  2. $C_i = F(IV + (i-1)) \oplus M_i$
- Dec($CT = (IV, C_1, \dots, C_T)$): $M_i = C_i \oplus F(IV + (i-1))$

The following Lemma is the main idea of the security proof,

Lemma 21.1. *Suppose that there exists a polynomial time algorithm, $\mathcal{A}$, such that*

$$Pr[\mathcal{A} \text{ succeeds in IND-CPA for CTR Mode }] - Pr[\mathcal{A} \text{ succeeds in IND-CPA for } system'] = \varepsilon$$

then there exists a polynomial time algorithm, $\mathcal{B}$ such that

$$Pr[\mathcal{B} \text{ outputs '1' } | \text{ oracle to PRP }] - Pr[\mathcal{B} \text{ outputs '1' } | \text{ oracle to TRP }] = \varepsilon.$$

Proof. Suppose that there exists an algorithm $\mathcal{A}$ with the properties given by the Lemma. We will use $\mathcal{A}$ to create the algorithm $\mathcal{B}$, as follows,

Algorithm 2 $\mathcal{B}$

$\mathcal{B}$ gets oracle access to $\mathcal{O}$ (either a PRP or a TRP)
$\mathcal{B}$ runs $\mathcal{A}$
while Query Phase **do**
  To encrypt $M = M_1 \dots M_T$, choose IV, set $C_i = \mathcal{O}(IV + (i-1)) \oplus M_i$
end while
while Challenge Phase **do**
  $\mathcal{A}$ gives m_0 and m_1.
  $\mathcal{B}$ flips a coin $\gamma \in \{0,1\}$
  $\mathcal{B}$ encrypts m_γ using the oracle and sends the result to $\mathcal{A}$
end while
while Query Phase II **do**
  To encrypt $M = M_1 \dots M_T$, choose IV, set $C_i = \mathcal{O}(IV + (i-1)) \oplus M_i$
end while
$\mathcal{A}$ outputs γ'
$\mathcal{B}$ outputs '1' if $\gamma = \gamma'$, '0' otherwise

The Lemma follows directly from our definition of $\mathcal{B}$. $\mathcal{B}$ will succeed in the PRP game exactly when $\mathcal{A}$ succeeds in distinguishing between the real CTR mode, *system*, and the truly random CTR mode, *system*$'$. □

Now we must show that *system*$'$ is secure. Consider querying on the zero vector, $\vec{0}$. The ciphertext response would be,

$$IV, F(IV) \oplus \vec{0}, F(IV+1) \oplus \vec{0}, \dots, F(IV+T) \oplus \vec{0}$$

Thus, by querying on the zero vector, we can see the output of F for $T+1$ different values. Then the challenge ciphertext will have the form,

$$IV^*, F(IV^*) \oplus M_{b_1}, F(IV^*+1) \oplus M_{b_2}, \ldots, F(IV^*+T) \oplus M_{b_T}$$

So, clearly, we don't want any of the values from IV^* to IV^*+T to be known to the attacker. Let's find the probability that there will be overlap in the initialization vector values from the queries and from the challenge ciphertext. In other words, the probability that the attacker will have any chance of breaking the system. For Q queries each of T blocks, we have

$$Pr[\text{ overlap }] \approx \frac{T \cdot Q}{2^l}.$$

Since there are 2^l total values in the range of F and $Q \cdot T$ of these will be known to the attacker. In actuality, the probability is slightly higher than this, but it is clear that these factors will cause the success probability of the attacker to be negligible. Hence, the system will be secure.

22 Message Authentication Codes (MAC)

Now we will see a new type of cryptographic protocol, appropriately called a message authentication code (MAC), that is used for message authentication. MACs consist of the three algorithms,

- $\text{Gen}(n) \rightarrow K$, where n is the security parameter and K is the key.
- $\text{MAC}(K, M) \rightarrow \sigma$, where M is a message and σ is the "tag" or "signature".
- $\text{Verify}(K, M, \sigma) \rightarrow$ { TRUE, FALSE } or $\{0, 1\}$

Similar to how we did for encryption schemes, we now define correctness and security properties for MACs.

Correctness

For any key, K, output by Gen,

$$\text{Verify}(K, M, Mac(K, M, \sigma)) = \text{TRUE}.$$

Security

1. Challenger runs $\text{Gen} \rightarrow K$
2. (Query Phase) Attacker gets oracle access to $\text{MAC}(K, \cdot)$ for queries $m_1, \ldots, m_Q$
3. Attacker gives a forgery, σ^*, on m^*, where $m^* \neq m_i$ from the query phase.

We say that an attacker succeeds in breaking the MAC scheme if

$$\text{Verify}(K, m^*, \sigma^*) = \text{TRUE}.$$

23 MAC Security

The notion of security that we will consider for MAC schemes is called Existential Unforgeability under Chosen Message Attack. This is the only generally accepted notion of security for MAC systems. The security game proceeds as follows,

1. Challenger runs $\text{Gen} \rightarrow K$
2. (Query Phase) Attacker gets oracle access to $\text{MAC}(K, \cdot)$ for queries $m_1, \ldots, m_Q$. Challenger returns $\sigma_1 = \text{MAC}(K, m_1), \ldots, \sigma_Q = \text{MAC}(K, m_Q)$, where Q is a polynomial in n. Note: The queries are adaptive. Attacker gets to see the response to the previous query before sending the next query.
3. (Forgery Attempt) Attacker gives a forgery, σ^*, on m^*.

We say that an attacker succeeds in breaking the MAC scheme if

1. $\text{Verify}(K, m^*, \sigma^*) = \text{TRUE}(or `1'')$, AND
2. $\forall 1 \leq i \leq Q, m^* \neq m_i$.

We say that a MAC scheme is secure if for all polynomial time attackers, $\mathcal{A}$,

$$Pr[\mathcal{A} \text{ succeeds}] \leq negl(n).$$

Now we will see our first secure MAC scheme. Like we did for our first secure encryption scheme, we will build our first secure MAC scheme from a PRF.

24 Secure MAC Scheme

Our MAC scheme will be over the set of l-bit messages. Suppose we have a PRF, $F_K : \{0,1\}^l \rightarrow \{0,1\}^z$, where the key, K, is n bits long. Our MAC scheme is defined,

- $\text{Gen}(n)$: Choose a PRF key, $K \in \{0,1\}^n$
- $\text{MAC}(K, M \in \{0,1\}^l)$: $\sigma = F_K(M)$
- $\text{Verify}(K, M, \sigma \in \{0,1\}^z)$: $\sigma = F_K(M)$

Now we will give a proof of security very similar to the proof of security we saw for the PRF-based encryption scheme.

24.1 Security

We are going to assume the security of the PRF and try to translate that security to existential unforgeability under chose message attack security for our MAC scheme. Just like we did for the encryption scheme, we will replace the PRF with a truly random function and compare the security of the two schemes. Then we will show that the scheme with the truly random function is information theoretically secure.

System' –"Truly Random MAC"

Here is the definition of the truly random MAC system that we will use to prove the security of our real system,

- Gen(n): Choose a truly random function, $F : \{0,1\}^l \rightarrow \{0,1\}^z$
- MAC($K, M \in \{0,1\}^l$): $\sigma = F(M)$
- Verify($K, M, \sigma \in \{0,1\}^z$): $\sigma = F(M)$

Remember, this scheme is completely impractical because a truly random function requires a huge $2^l z$-bit table. However, we can still use this scheme in our proof.

In general, if the security of a cryptographic system is based on a PRF or a PRP then the proof of security will almost always depend on a system using a truly random function in place of the psuedorandom function.

Lemma 24.1. *Suppose that F_K is a secure PRF, then for any polynomial time attacker, $\mathcal{A}$,*

$$Pr[\mathcal{A} \textit{ succeeds in MAC scheme}] - Pr[\mathcal{A} \textit{succeeds in System'}] = negl(n).$$

Let $Pr[\mathcal{A}_{MAC}]$ be the probability that an attacker, $\mathcal{A}$, succeeds in our original MAC scheme and let $Pr[\mathcal{A}_{SYS'}]$ be the probability that $\mathcal{A}$ succeeds in System'.

Claim 24.2. Suppose that $Pr[\mathcal{A}_{MAC}] - Pr[\mathcal{A}_{SYS'}] = \varepsilon$ then there exists a polynomial time algorithm $\mathcal{B}$ such that,

$$Pr[\mathcal{B}^{F_K(\cdot)} = 1] - Pr[\mathcal{B}^{F(\cdot)} = 1] = \varepsilon,$$

where F_K is the PRF used in our MAC scheme and F is a truly random function.

This claim immediately gives rise to the lemma. To prove this claim, we will construct the algorithm $\mathcal{B}$ that can be used to break the PRF, F_K.

Algorithm 3 $\mathcal{B}$

Proof.
1: $\mathcal{B}$ gets oracle access to $\mathcal{O}(\cdot)$ (either a PRf or a TRF)
2: $\mathcal{B}$ runs $\mathcal{A}$
3: **while** Query Phase **do**
4: $\quad\mathcal{A}$ asks for MACs on $M_1, \ldots, M_Q$
5: $\quad$Give $\sigma_i = \mathcal{O}(M_i)$ to $\mathcal{A}$
6: **end while**
7: $\mathcal{A}$ gives attempted forgery, σ^* on M^*
8: $\mathcal{B}$ outputs '1' if and only if $\mathcal{O}(M^*) = \sigma^*$ AND $\forall 1 \leq i \leq Q, M^* \neq M_i$

Suppose that $\mathcal{O}(\cdot)$ is a PRF oracle, then $\mathcal{A}$ will be playing the original MAC game, so $\mathcal{B}$ will output '1' when $\mathcal{A}$ is successful. On the other hand, if $\mathcal{O}(\cdot)$ is a TRF, then $\mathcal{A}$ will be playing the System' game , so $\mathcal{B}$ will output '1' when $\mathcal{A}$ is successful in this game. Hence,

$$Pr[\mathcal{B}^{F_K(\cdot)} = 1] - Pr[\mathcal{B}^{F(\cdot)} = 1] = Pr[\mathcal{A}_{MAC}] - Pr[\mathcal{A}_{SYS'}] = \varepsilon.$$

□

The lemma follows directly from this claim. Given the lemma, we leave it as an exercise to show that System' is information theoretically secure. The argument we will be very similar to the information theoretic argument for the security of the truly random encryption scheme that we saw in a previous lecture.

25 Chosen Ciphertext Security

So far, we have discussed the security of encryption systems under chosen plaintext attacks, and we have defined the notion of indistinguishability under such attacks. Today, we will consider a different type of attacked, called a chosen ciphertext attack and we will define a new notion of security against this type of attacks, called indistinguishability under chose ciphertext attacks (IND-CCA).

The IND-CCA security game is defined,

1. Challenger runs KeyGen
2. (Query Phase I) Attacker is given access to two oracles, $\text{Enc}(K, \cdot)$ and $\text{Dec}(K, \cdot)$.
3. (Challenge Phase) Attacker produces two messages m_0 and m_1. The challenger returns the challenge ciphertext $c^* = \text{Enc}(K, m_b)$.
4. (Query Phase II) Same as Query Phase I except cannot query the decryption oracle on c^*.
5. Attacker outputs b'

We define the advantage of an attacker $\mathcal{A}$ in the IND-CCA security game to be

$$Adv_{\mathcal{A}} = Pr[b' = b] - \frac{1}{2}.$$

We say that an encryption scheme is IND-CCA secure if for any polynomial time attacker, $\mathcal{A}$,

$$Adv_{\mathcal{A}} = negl(n).$$

26 IND-CPA vs. IND-CCA

Our previous definition of security, IND-CPA security, intuitively captured the notion that an attacker can see specific messages sent between the two communicating parties. Now, we define this stronger notion of security, where the attacker can inject messages into the stream and also see how they are decrypted. This is called a chosen ciphertext attack. If an encryption scheme is secure under chosen ciphertext attacks then we say that it is IND-CCA secure.

The IND-CCA security game is the same as the IND-CPA game, except that now the attacker has access to a decryption oracle in addition to the encryption oracle. However, notice that the attacker is not allowed to query on the challenge ciphertext, or else it would be impossible to have a scheme that is IND-CCA secure.

We consider this definition of security to be very robust and even overly cautious. Note that IND-CCA security includes IND-CPA security because an attacker can always just choose not to use the decryption oracle. Is converse true, that if a scheme is IND-CPA secure it must be IND-CCA secure?

Suppose we have an encryption system, $\Pi = (G, E, D)$, where Π is IND-CPA secure. Then we can construct $\Pi' = (G', E', D') = (G, E', D)$, where $E'(M)$:

1. Choose random bit $a \in \{0, 1\}$
2. Return $CT = (E(m), a)$.

First, we want to argue that this scheme is IND-CPA secure. This is true because the original scheme is IND-CPA secure and giving out a does not actually leak any new information. However, suppose we get the challenge ciphertext,

$$CT^* = (E(m_b, a))$$

then an attacker can query,

$$D(K, (E(m_b, \bar{a})) = m_b$$

So, Π' is not IND-CCA secure. Hence, it is not the case that if an encryption scheme is IND-CPA secure then it must be IND-CCA secure.

By definition, we know that if an encryption scheme is IND-CCA secure then it is IND-CPA secure. This is equivalent to the contrapositive statement: if an encryption scheme is not IND-CPA secure then it is not IND-CCA secure. In other words, if there exists an efficient IND-CPA attacker then there exists an efficient IND-CCA attacker. In fact, this attacker is exactly the same, because it can just choose not to use the decryption oracle in the IND-CCA game.

26.1 CCA Security of the PRF Scheme

The new encryption scheme we have created may seem like a toy example, but are there any schemes that we have seen so far that are IND-CPA secure, but not IND-CCA secure? Let's consider our PRF encryption scheme, if F_K is a PRF with key K,

- Gen - Choose K as a PRF key
- $\text{Enc}(K, M)$ - Choose a random r and return $c = (r, F_K(r) \oplus M)$
- $\text{Dec}(K, c = (c_1, c_2))$ - Return $m = c_2 \oplus F_K(c_1)$.

Remember, this scheme is IND-CPA secure. Is it IND-CCA secure?

Suppose that we get the challenge ciphertext $c^* = (r^*, m_b \oplus F_K(r^*))$. Then we can query the decryption oracle on $c = (r^*, \vec{0})$, which will give us $F_K(r^*)$. Then, we can use this to get $F_K(r^*) \oplus c_2^* = m_b$! So, this scheme is NOT IND-CCA secure.

In fact, we do not even need to use the zero message, we can use any string as c_2 in our query and then XOR the result with the result of $\text{Dec}(K, c)$ and c_2^* to get back to the challenge message. So, this is a real scheme that is IND-CPA secure but not IND-CCA secure.

This system breaks down against CCAs because in the scheme there is a negligible chance of getting the same randomness, but with CCAs the attacker can choose the randomness.

So, now let's try to build an IND-CCA secure scheme.

27 IND-CCA Secure Scheme

When building a CCA secure scheme, we want to make sure that an attacker will not be able to inject valid messages to the decryption oracle. So, we use a MAC to authenticate messages to make sure that they are not sent from an attacker. Recall the MAC security game,

1. Challenger runs $\text{Gen} \rightarrow K$
2. (Query Phase) Attacker gets oracle access to $\text{MAC}(K, \cdot)$ for queries $m_1, \ldots, m_Q$. Challenger returns $\sigma_1 = \text{MAC}(K, m_1), \ldots, \sigma_Q = \text{MAC}(K, m_Q)$, where Q is a polynomial in n. Note: The queries are adaptive. Attacker gets to see the response to the previous query before sending the next query.
3. (Forgery Attempt) Attacker gives a forgery, σ^*, on m^*.

Let's try a few different ways of incorporating the MAC scheme into our encryption scheme and see how it effects the CCA security.

27.1 Mac and Encrypt

First, let's just try to both encrypt and sign the message.

- Gen: Let $K_{CCA} = K_{CPA}, K_{\text{MAC}}$
- $Enc_{CCA}(K_{CCA}, M)$: $(c_1 = Enc_{CPA}(K_{CPA}, M), c_2 = \text{MAC}(K_{\text{MAC}}, M))$
- $Dec_{CCA}(K_{CCA}, C)$:
  1. $X = D_{CPA}(K_{CPA}, c_1)$
  2. if $\text{Verify}(K_{\text{MAC}}, X, c_2)$ output X
  3. else $\perp$

Is this scheme enough to give us IND-CCA security? Consider the following attack. We can query

$$Enc_{CCA}(m_0) = (Enc_{CPA}(m_0), \text{MAC}(m_0))$$

then during the challenge phase, we give m_0 and m_1 to the challenger and receiver back,

$$c^* = (Enc_{CPA}(m_b), \text{MAC}(m_b)).$$

In the second phase, we can query $c = (Enc_{CPA}(m_b) = c_1^*, \text{MAC}(m_0))$

If the MAC scheme is randomized, then $\text{MAC}(m_0) \neq c_2^*$, so the query is okay and the decryption oracle's response will tell us which message has been encrypted. If the MAC scheme is not randomized, then it could be the case that it just gives away the message. Also, we could use the same attack we did before when we created the Π' scheme, by just adding a random bit to the ciphertext. So, this scheme will not be IND-CCA secure. Let's try a different way of using the MAC scheme in the encryption scheme.

27.2 Encrypt then MAC

Let's try encrypting the message first and then signing it,

- Gen: Let $K_{CCA} = K_{CPA}, K_{\text{MAC}}$
- $Enc_{CCA}(K_{CCA}, M)$: $(c_1 = Enc_{CPA}(K_{CPA}, M), c_2 = \text{MAC}(K_{\text{MAC}}, c_1))$
- $Dec_{CCA}(K_{CCA}, C)$:
  1. if $\text{Verify}(K_{\text{MAC}}, c_1, c_2)$ output $m = Dec_{CPA}(K_{CPA}, c_1)$
  2. else $\perp$

Intuitively, in this new scheme we cannot add anything to the ciphertext without breaking the MAC verification. Next lecture we will prove that this system is IND-CCA secure.

28 Encrypt-Then-MAC

Now, we will prove that the encryption system we introduced last lecture is IND-CCA secure. Recall that the system was defined as follows,

- Gen: Let $K_{CCA} = K_{CPA}, K_{\text{MAC}}$
- $Enc_{CCA}(K_{CCA}, M)$: $(c_1 = Enc_{CPA}(K_{CPA}, M), c_2 = \text{MAC}(K_{\text{MAC}}, c_1))$
- $Dec_{CCA}(K_{CCA}, C)$:
  1. if $\text{Verify}(K_{\text{MAC}}, c_1, c_2)$ output $m = Dec_{CPA}(K_{CPA}, c_1)$
  2. else $\perp$

We want to prove that this scheme is secure. In order to prove the scheme is secure, we will give our first proof using a hybrid proof technique.

First we will create a different security game, that will be used in our hybrid proof.

28.1 DecrpytListGame

This game will define a different notion of security that is specific to our encryption scheme and we will show that it is related to IND-CCA security in our proof. The game is defined,

1. Challenger runs KeyGen
2. (Query Phase I) For each query M_i to the encryption oracle, return the ciphertext $c_i = (c_{i,A}, c_{i,B})$ and add $(M_i, c_{i,A})$ to a list.
3. (Query Phase I) For each query $c_i = (c_{i,A}, c_{i,B})$ to the decryption oracle if $c_{i,A}$ is on the list and c_i verifies then return the corresponding M, else return $\perp$.
4. (Challenge Phase) Attacker produces two messages m_0 and m_1. The challenger returns the challenge ciphertext $c^* = \text{Enc}(K, m_b)$.
5. (Query Phase II) Same as Query Phase I except cannot query the decryption oracle on c^*.
6. Attacker outputs b'

Intuitively, in this new game, an attacker can only make decryption queries for ciphertexts that he has already received. Hence, it should be harder to break this game than the CCA game. However, we want to show that if the MAC scheme is secure then it isn't any easier to break the CCA game than the DecryptListGame. Once we have shown this, then we will argue that the DecryptListGame is close to the IND-CPA security game since the decryption oracle is essentially useless.

Lemma 28.1. *If the MAC scheme is secure then for any efficient algorithm, $\mathcal{A}$,*

$$Pr[\mathcal{A}^{CCA}_{succ}] - Pr[\mathcal{A}^{DListGame}_{succ}] = negl(n).$$

Proof. Suppose that there exists and efficient algorithm, $\mathcal{A}$ such that $Pr[\mathcal{A}^{CCA}_{succ}] - Pr[\mathcal{A}^{DListGame}_{succ}] = \varepsilon$, then there exists and algorithm, $\mathcal{B}$, such that the advantage of $\mathcal{B}$ in the MAC game is ε/Q, where Q is the number of decryption queries made by $\mathcal{A}$. Let 'new' be the event that a $c_{i,A}$ query verified that was not on the list of encryption queries, then for the CCA game

$$Pr[\mathcal{A}^{CCA}_{succ}] = Pr[\mathcal{A}^{CCA}_{succ}|new]Pr[new] + Pr[\mathcal{A}^{CCA}_{succ}|n\bar{e}w]Pr[n\bar{e}w]$$

Likewise, for the DecryptListGame,

$$Pr[\mathcal{A}^{DListGame}_{succ}] = Pr[\mathcal{A}^{DListGame}_{succ}|new]Pr[new] + Pr[\mathcal{A}^{DListGame}_{succ}|n\bar{e}w]Pr[n\bar{e}w]$$

However, the games are identical when $n\bar{e}w$ occurs, so $Pr[\mathcal{A}^{CCA}_{succ}|n\bar{e}w] = Pr[\mathcal{A}^{DListGame}_{succ}|n\bar{e}w]$ and, of course $Pr[n\bar{e}w]$ does not change. So, when we subtract the two values from ea14h other the second term in each will cancel. So, we get

$$\begin{aligned}\varepsilon &= (Pr[\mathcal{A}^{CCA}_{succ}|new] - Pr[\mathcal{A}^{DListGame}_{succ}|new])Pr[new] \\ &\leq 1 \cdot Pr[new]\end{aligned}$$

Algorithm 4 $\mathcal{B}$

Given oracle access to MAC and Verify. $K_{CPA} = Gen_{\text{CPA}}()$.
Runs $\mathcal{A}$.
For query $Enc_{\text{CCA}}(M)$, responds $c = (c_A = Enc_{\text{CPA}}(K_{\text{CPA}}, M), c_B = \text{OracleMAC}(c_A))$
For query $Dec_{\text{CCA}}((c_A, c_B))$, use VerifyOracle$(c_A, c_B)$.
If at any point a decryption query verifies and is not on the list, return it as a forgery.
if VerifyOracle(c_A, c_B) = 'FALSE' **then**
  return $\perp$
else
  return $Dec_{\text{CPA}}(K_{\text{CPA}}, c_A)$
end if

So, $Pr[new] \geq \varepsilon$. Now, we will define an algorithm, $\mathcal{B}$ that breaks the MAC scheme with probability ε/Q. We define the algorithm as follows,

Now, we must analyze $\mathcal{B}$ to see if it succeeds with non-negligible probability, ε/Q.

Case 0: Decryption query does not verify.

Case 1: Query verifies but is on the list.

Case 2: Query verifies and is not on the list (so, we can use it as a forgery).

Considering these cases, it is easy to see that $\mathcal{B}$ will succeed with probability ε/Q. We leave this as an exercise. □

So, if the MAC scheme was secure then any algorithm successful against the CCA scheme will be successful against the DecryptListGame. However, anything successful in the DecryptListGame will successful in the IND-CPA game, but we assumed that this scheme is secure. So, no efficient algorithm will be successful in breaking the CCA scheme with non-negligible probability. We call this a hybrid proof technique because the DecryptListGame is between CCA and CPA.

29 Introduction

These notes are intended to provide a brief survey of some basic concepts in a branch of algebra called group theory. Many claims made in these notes are stated without proof. Consult your cryptography textbook or a textbook on abstract algebra for reference.

Here we review some basic notation. We denote the set of integers by $\mathbb{Z}$, the set of natural numbers (non-negative integers) by $\mathbb{N}$, and the set of positive integers by $\mathbb{Z}_+$; that is,

$$\begin{aligned}\mathbb{Z} &= \{\dots, -3, -2, -1, 0, 1, 2, 3, \dots\},\\ \mathbb{N} &= \{0, 1, 2, 3, \dots\},\\ \mathbb{Z}_+ &= \{1, 2, 3, \dots\}.\end{aligned}$$

The set of rational numbers is denoted by $\mathbb{Q}$; that is,

$$\mathbb{Q} = \left\{ \frac{a}{b} \,:\, a, b \in \mathbb{Z} \text{ and } b \neq 0 \right\}.$$

We also denote the set of real numbers by $\mathbb{R}$ and the set of complex numbers by $\mathbb{C}$. We write $\{0,1\}^*$ to denote the set of all strings in which each character is either 0 or 1. The symbol $\|$ denotes string concatenation; that is, for all strings x and y, we write $x\|y$ to denote the result of appending y to x. The **identity function** $\mathrm{id}_A : A \to A$ on a set A is defined as the function that maps every element of A to itself; that is,

$$\mathrm{id}_A(x) := x \quad \text{for all } x \in A.$$

30 Sets with an operation

Before we define groups, it may be helpful to consider sets with an operation in general.[1]

Definition 30.1. Let S be a set with an operation $\circ$ defined on any two elements of S. We say that S is **closed** under $\circ$ if

$$a \circ b \in S \quad \text{for all } a, b \in S.$$

We say that $\circ$ is **associative** if

$$(a \circ b) \circ c = a \circ (b \circ c) \quad \text{for all } a, b, c \in S.$$

We say that $\circ$ is **commutative** if

$$a \circ b = b \circ a \quad \text{for all } a, b \in S.$$

Remark: The members of S need not be real numbers. We will see examples of this later. The operations discussed in these notes are all associative. However, they may or may not be commutative.

Notation: If the operation $\circ$ is associative, then we can omit the parentheses and write $a \circ b \circ c$ instead of $(a \circ b) \circ c$ or $a \circ (b \circ c)$. Similar notation is used with four or more elements.

Definition 30.2. Let S be a set closed under an associative operation $\circ$. An **identity element** of S under $\circ$ is an element $e \in S$ such that

$$a \circ e = a = e \circ a \quad \text{for all } a \in S.$$

We prove next that S cannot have more than one identity element.

Theorem 30.3. *Let S be a set closed under an associative operation $\circ$. The set S contains at most one identity element under $\circ$.*

Proof. If e and e' are both identity elements of S, then $e = e \circ e' = e'$ by definition. □

[1] Readers already familiar with algebra will recognize that this section is about *semigroups* and *monoids*.

Definition 30.4. Let S be a set closed under an associative operation $\circ$ with identity element e. An **inverse** of an element $a \in S$ under $\circ$ is an element $b \in S$ such that

$$a \circ b = e = b \circ a.$$

A useful property of elements with an inverse is that they obey the left and right cancellation laws.

Theorem 30.5. *Let S be a set closed under an associative operation $\circ$ with identity element e. Let $c, x, y \in S$. If c has an inverse under $\circ$, then*

$$c \circ x = c \circ y \quad \textit{implies} \quad x = y \quad \textit{(left cancellation)}$$

and

$$x \circ c = y \circ c \quad \textit{implies} \quad x = y \quad \textit{(right cancellation).}$$

Proof. Suppose c has an inverse $b \in S$. If $c \circ x = c \circ y$, then

$$x = e \circ x = (b \circ c) \circ x = b \circ (c \circ x) = b \circ (c \circ y) = (b \circ c) \circ y = e \circ y = y.$$

If $x \circ c = y \circ c$, then

$$x = x \circ e = x \circ (c \circ b) = (x \circ c) \circ b = (y \circ c) \circ b = y \circ (c \circ b) = y \circ e = y.$$

□

Remark: If $\circ$ is not commutative, then we cannot use cancellation to deduce $x = y$ from $c \circ x = y \circ c$.

We now prove that an element of S cannot have more than one inverse.

Theorem 30.6. *Let S be a set closed under an associative operation $\circ$ with identity element e. Every element of S has at most one inverse under $\circ$.*

Proof. Let $a \in S$, and let $b, b' \in S$ both be inverses of a. Then $a \circ b = e = a \circ b'$ by definition, so $b = b'$ by left cancellation. □

Notation: For brevity, we shall write $(S, \circ, e)$ to mean the set S with operation $\circ$ and identity element e.

We now give two examples of sets with non-commutative operations whose members are not real numbers.

Example 30.7. Consider the set of strings $\{0,1\}^*$ under the string concatenation operation $\|$. The identity element e of $\{0,1\}^*$ is the empty string. We write $\left(\{0,1\}^*, \| , e\right)$ to mean the set $\{0,1\}^*$ with operation $\|$ and identity element e. The only member of $\{0,1\}^*$ with an inverse is the empty string.

Example 30.8. Let A be a set, and let S be the set of all functions from A to A. Let $\circ$ denote function composition. The identity element of S under $\circ$ is the identity function id_A. We write $(S, \circ, \mathrm{id}_A)$ to mean the set S with operation $\circ$ and identity element id_A. Not every member of S has an inverse. However, the permutations all have inverses.

We define repeated application of an operation $\circ$ as follows.

Definition 30.9. Let S be a set closed under an associative operation $\circ$ with identity element e. Let $a \in S$. We recursively define

$$a^0 := e \quad \text{and} \quad a^{m+1} := a^m \circ a \quad \text{for all } m \in \mathbb{N}.$$

Remark: The intuitive meaning of a^m is the following:

$$a^m = \underbrace{a \circ \cdots \circ a}_{m \text{ times}}.$$

We can prove by induction that repeated application of $\circ$ behaves as one would expect.

Theorem 30.10. *Let S be a set closed under an associative operation $\circ$ with identity element e. Let $a \in S$. Then repeated application of the operation $\circ$ obeys the following rules:*

$$a^m \in S, \qquad a^{m+n} = a^m \circ a^n, \qquad (a^m)^n = a^{mn} \qquad \textit{for all } m, n \in \mathbb{N}.$$

31 Groups

In this section, we define groups and discuss some of their basic properties.

Definition 31.1. A **group** is a set G along with an operation $\circ$ defined on any two elements of G and an element $e \in G$ such that the following properties hold:

closure: $a \circ b \in G$ for all $a, b \in G$.

associativity: $(a \circ b) \circ c = a \circ (b \circ c)$ for all $a, b, c \in S$.

identity element: $a \circ e = a = e \circ a$ for all $a \in S$.

existence of inverses: For every $a \in G$, there exists an element $a^{-1} \in G$ such that

$$a \circ a^{-1} = e = a^{-1} \circ a.$$

Definition 31.2. A group $(G, \circ, e)$ is called **abelian** or **commutative** if

$$a \circ b = b \circ a \quad \text{for all } a, b \in G.$$

Remark: The group operation $\circ$ is assumed to be associative, but may or may not be commutative. There exist non-commutative groups.

The next theorem follows immediately from the definition of a group and Theorems 30.3 and 30.6.

Theorem 31.3. *Let $(G, \circ, e)$ be a group. Then:*

1. *The element e is the unique identity element of G under $\circ$.*
2. *For each $a \in G$, the element a^{-1} is the unique inverse of a under $\circ$.*

Example 31.4. We illustrate the concept of a group with a few examples:

- $(\mathbb{Z}, +, 0)$, $(\mathbb{Q}, +, 0)$, $(\mathbb{R}, +, 0)$, and $(\mathbb{C}, +, 0)$ are commutative groups.
- $\big(\mathbb{Q} \setminus \{0\}, \cdot, 1\big)$, $\big(\mathbb{R} \setminus \{0\}, \cdot, 1\big)$, and $\big(\mathbb{C} \setminus \{0\}, \cdot, 1\big)$ are commutative groups.
- The set $\mathbb{Z} \setminus \{0\}$ is not a group under integer multiplication.
- The set of vectors in $\mathbb{R}^n$ is a commutative group under vector addition with the zero vector as the identity element.
- The set of $m \times n$ matrices with real entries is a commutative group under matrix addition with the matrix of all zeros as the identity element.
- The set of invertible $n \times n$ matrices with real entries is a non-commutative group under matrix multiplication with the $n \times n$ identity matrix as the identity element.
- The set of permutations on a set A is a non-commutative group under function composition with the identity function id_A as the identity element.

The next theorem states some basic rules for calculating inverses.

Theorem 31.5. *Let $(G, \circ, e)$ be a group. Then:*

1. *$e^{-1} = e$.*
2. *$(a^{-1})^{-1} = a$ for all $a \in G$.*
3. *$(a \circ b)^{-1} = b^{-1} \circ a^{-1}$ for all $a, b \in G$.*

Proof. Since $e \circ e = e$ by definition, we know that e is an inverse of e. But e^{-1} is the only inverse of e. Therefore, $e^{-1} = e$. Let $a \in G$. Since $a \circ (a^{-1}) = e = (a^{-1}) \circ a$ by definition, we know that a is an inverse of a^{-1}. But $(a^{-1})^{-1}$ is the only inverse of a^{-1}. Therefore, $(a^{-1})^{-1} = a$. Now, let $a, b \in G$. Observe that

$$(a \circ b) \circ (b^{-1} \circ a^{-1}) = \big((a \circ b) \circ b^{-1}\big) \circ a^{-1} = \big(a \circ (b \circ b^{-1})\big) \circ a^{-1} = (a \circ e) \circ a^{-1} = a \circ a^{-1} = e.$$

and

$$(b^{-1} \circ a^{-1}) \circ (a \circ b) = b^{-1} \circ \big(a^{-1} \circ (a \circ b)\big) = b^{-1} \circ \big((a^{-1} \circ a) \circ b\big) = b^{-1} \circ (e \circ b) = b^{-1} \circ b = e.$$

Hence, $b^{-1} \circ a^{-1}$ is an inverse $a \circ b$. But $(a \circ b)^{-1}$ is the only inverse of $a \circ b$. Thus, $(a \circ b)^{-1} = b^{-1} \circ a^{-1}$. $\square$

Remark: If the operation $\circ$ is not commutative, then $b^{-1} \circ a^{-1}$ may be different from $a^{-1} \circ b^{-1}$. For example, if A and B are invertible matrices, then $B^{-1}A^{-1}$ is the inverse of AB, but $A^{-1}B^{-1}$ may be a different matrix from $B^{-1}A^{-1}$.

Recall that we defined a^m as follows:

$$a^0 := e \quad \text{and} \quad a^{m+1} := a^m \circ a \quad \text{for all } m \in \mathbb{N}.$$

Since every member of a group has an inverse, we can extend this definition the case when $m < 0$.

Definition 31.6. Let $(G, \circ, e)$ be a group. Let $a \in G$. We define

$$a^{-m} := \left(a^{-1}\right)^m \quad \text{for all } m \in \mathbb{Z}_+.$$

We can prove by induction that repeated application of $\circ$ under this extended definition obeys the rules one would expect.

Theorem 31.7. *Let $(G, \circ, e)$ be a group. Let $a \in G$. Then,*

$$a^m \in G \quad \textit{and} \quad a^{-m} = \left(a^{-1}\right)^m = (a^m)^{-1} \quad \textit{for all } m \in \mathbb{Z},$$

and

$$a^{m+n} = a^m \circ a^n \quad \textit{and} \quad (a^m)^n = a^{mn} \quad \textit{for all } m, n \in \mathbb{Z}.$$

Multiplicative and additive notation

Here we introduce two notations for writing groups. In multiplicative notation, the group operation is denoted by $\cdot$, the identity element by 1, the inverse of a by a^{-1}, and m applications of the operation $\cdot$ to a by a^m. We often write ab instead of $a \cdot b$ for convenience. If the operation $\cdot$ is commutative, then we write $\frac{a}{b}$ or a/b for ab^{-1}. (This notation is avoided when $\cdot$ is not commutative, because it is not clear whether $\frac{a}{b}$ means ab^{-1} or $b^{-1}a$.)

In additive notation, the group operation is denoted by $+$, the identity element by 0, the inverse of a by $-a$, and m applications of the operation $+$ to a by ma. If the operation $+$ is commutative, then we write $a - b$ for $a + (-b)$. (Most authors use additive notation only for commutative operations.) These notations are summarized in the table below.

	multiplicative notation	additive notation
group operation	$a \cdot b$ or ab	$a + b$
identity element	1	0
inverse	a^{-1}	$-a$
repeated application	a^m	ma
operation and inverse	$\frac{a}{b}$ or a/b	$a - b$

Notational conventions: Below we describe some notational conventions commonly used in algebra.

- Additive notation is usually meant to signal to the reader that the operation is commutative. On the other hand, multiplicative notation does not connote commutativity.
- When the group operation and identity element are not specified, the usual convention is to assume multiplicative notation.
- We frequently need to work with two groups G and G' at the same time. It is common to use the same symbol $\cdot$ to denote both the operation on G and the operation on G'. Context determines which operation is meant. Similarly, the same symbol may be used to denote the identity elements of the two groups. Similar comments also apply when working with three or more groups.

We shall henceforth adopt the above conventions.

32 Modular arithmetic revisited

Let a and b be integers and n be a positive integer. Since $(a+b) \bmod n$ and $ab \bmod n$ are always integers between 0 and $n-1$, we can restrict ourselves to the set $\{0, \ldots, n-1\}$ when working modulo n. This observation motivates the following definition of $\mathbb{Z}_n$.

Definition 32.1. In this definition only, we write $(a+b)_{\mathbb{Z}}$ and $(a \cdot b)_{\mathbb{Z}}$ to respectively denote the integer sum and integer product of two integers a and b; this notation is used to avoid confusion with the addition and multiplication operations that we will define on $\mathbb{Z}_n$.

Let n be a positive integer. We define $\mathbb{Z}_n$ to be the set $\{0, \ldots, n-1\}$ with the following operations:

addition:	$a + b := (a+b)_{\mathbb{Z}} \bmod n$	for all $a, b \in \mathbb{Z}_n$.
multiplication:	$a \cdot b := (a \cdot b)_{\mathbb{Z}} \bmod n$	for all $a, b \in \mathbb{Z}_n$.

For convenience, we sometimes write ab instead of $a \cdot b$.

Remark: Notice that 0 is the identity element of $\mathbb{Z}_n$ under addition and 1 is the identity element of $\mathbb{Z}_n$ under multiplication.

Definition 32.2. We define $\mathbb{Z}_n^*$ as the set of all elements in $\mathbb{Z}_n$ with an inverse under multiplication; that is,

$$\mathbb{Z}_n^* := \big\{a \in \mathbb{Z}_n : ab = 1 = ba \text{ for some } b \in \mathbb{Z}_n\big\}.$$

With these definitions, we can restate some basic results of number theory in the language of algebra.

Theorem 32.3. *For all $n \in \mathbb{Z}_+$:*

1. *$(\mathbb{Z}_n, +, 0)$ is a commutative group.*
2. *$(\mathbb{Z}_n^*, \cdot, 1)$ is a commutative group.*

Remark: The set $\mathbb{Z}_n^*$ is not a group under addition, because $\mathbb{Z}_n^*$ does not contain 0.

Remark: We write the group $(\mathbb{Z}_n, +, 0)$ in additive notation and the group $(\mathbb{Z}_n^*, \cdot, 1)$ in multiplicative notation. For each $a \in \mathbb{Z}_n$, the inverse of a under addition is $n-a$, which is written $-a$ in additive notation. For all $m, a \in \mathbb{Z}_n$, the product of m and a coincides with the element of $\mathbb{Z}_n$ obtained by adding a to itself m times; hence, there is no ambiguity in using the same notation ma for both.

Example 32.4. In the table below, we compare congruence modulo 7 with arithmetic in $\mathbb{Z}_7$. In the left column, addition and multiplication are performed in $\mathbb{Z}$. In the right column, addition and multiplication are performed in $\mathbb{Z}_7$. Note that in the right column, -4 denotes the inverse of 4 under addition in $\mathbb{Z}_7$.

congruence modulo 7	arithmetic in $\mathbb{Z}_7$
$5 + 6 \equiv 11 \equiv 4 \pmod 7$	$5 + 6 = 4$
$1 - 5 \equiv -4 \equiv 3 \pmod 7$	$1 - 5 = -4 = 3$
$4 \cdot 5 \equiv 20 \equiv 6 \pmod 7$	$4 \cdot 5 = 6$

Example 32.5. Addition and multiplication in $\mathbb{Z}_n$ depend, of course, on the modulus n. Context is required to determine which operations are meant. The table below illustrates the difference between addition and multiplication in $\mathbb{Z}_{12}$ and addition and multiplication in $\mathbb{Z}_7$.

arithmetic in $\mathbb{Z}_{12}$	arithmetic in $\mathbb{Z}_7$
$5 + 6 = 11$	$5 + 6 = 4$
$1 - 5 = -4 = 8$	$1 - 5 = -4 = 3$
$4 \cdot 5 = 8$	$4 \cdot 5 = 6$

The following theorems are just restatements of results from number theory.

Theorem 32.6. *For all $n \in \mathbb{Z}_+$,*

$$\mathbb{Z}_n^* = \{a \in \mathbb{Z}_n : \gcd(a, n) = 1\}.$$

Theorem 32.7. *Let p be a prime. Then:*

1. $\mathbb{Z}_p^* = \{1, \ldots, p-1\}$.
2. $a^{p-1} = 1$ *for all* $a \in \mathbb{Z}_p^*$ *(Fermat's little theorem).*

Alternative definition of $\mathbb{Z}_n$: The definition of $\mathbb{Z}_n$ given above follows the one in your cryptography textbook. You may see another definition of $\mathbb{Z}_n$ in other books. Rather than define $\mathbb{Z}_n$ as $\{0, \ldots, n-1\}$, we can define $\mathbb{Z}_n$ as the set of equivalence classes of integers congruent modulo n. We would then define the sum and product of two equivalence classes. For further details, consult a textbook on abstract algebra.

33 Subgroups and generators

In this section, we discuss subsets of a group that are also groups themselves. Henceforth, if the operation and identity element of a group are not specified, then multiplicative notation should be assumed.

Definition 33.1. Let G be a group and H be a subset of G. We say that H is a **subgroup** of G if H is also a group under the same operation as G.

Remark: A subset of G that is a group under some other operation is not necessarily a subgroup of G.

Remark: Every group G has at least two subgroups: $\{1\}$ and G itself. These subgroups are called the **trivial subgroups** of G.

An alternative characterization of subgroups is the following.

Theorem 33.2. *Let G be a group. A subset H of G is a subgroup of G if and only if H satisfies the following three conditions:*

1. $1 \in H$ *(contains the identity element).*
2. $ab \in H$ *for all* $a, b \in H$ *(closure under the group operation).*
3. $a^{-1} \in H$ *for all* $a \in H$ *(closure under inverses).*

If H is non-empty, then conditions 2 and 3 above suffice to show that H is a subgroup of G.

Remark: Suppose that H satisfies conditions 2 and 3 of Theorem 33.2. If H contains some element a, then $a^{-1} \in H$ (by condition 3) and hence $1 = aa^{-1} \in H$ (by condition 2). Thus, condition 1 is redundant when the subset H is non-empty.

Example 33.3. We illustrate the concept of subgroups with a few examples below:

- The set $\{0, 2, 4\}$ is a subgroup of $(\mathbb{Z}_6, +, 0)$.
- The set $\{1, 2, 4\}$ is a subgroup of $(\mathbb{Z}_7^*, \cdot, 1)$.
- The set $\{0, 1, 5\}$ is not a subgroup of $(\mathbb{Z}_6, +, 0)$, because it is not closed under addition.
- The set $\{1, 3, 5\}$ is not a subgroup of $(\mathbb{Z}_7^*, \cdot, 1)$, because it is not closed under multiplication.

Definition 33.4. Let G be a finite group, that is, a group with finitely many elements. The **order** of G is defined as the size $|G|$ of the set G.

Definition 33.5. Let G be a finite group and let $g \in G$. The **subgroup of G generated by** g is the set $\langle g \rangle$ defined by

$$\langle g \rangle := \{g^m : m \in \mathbb{N}\}.$$

Remark: Even though we call $\langle g \rangle$ *the subgroup of G generated by g*, we must still verify that $\langle g \rangle$ actually is a subgroup of G.

Theorem 33.6. *Let G be a finite group. For each $g \in G$, the set $\langle g \rangle$ is a subgroup of G, and furthermore, the order of $\langle g \rangle$ is the smallest positive integer N such that $g^N = 1$.*

Proof. Let $g \in G$. Note that $\langle g \rangle$ is a subset of G. Since G is a finite set, every subset of G, including $\langle g \rangle$, must be a finite set as well. Therefore, the terms of the sequence $(g^0, g^1, g^2, g^3, \dots)$ cannot be all distinct. In other words, there exist $k, \ell \in \mathbb{N}$ such that $k < \ell$ and $g^k = g^\ell$. Then $\ell - k > 0$ and $g^{\ell-k} = 1$, so there is at least one positive integer n such that $g^n = 1$. Let N be the smallest positive integer such that $g^N = 1$. We claim that the set $\{g^0, \dots, g^{N-1}\}$ contains exactly N elements. Suppose, to the contrary, that there exist $i, j \in \mathbb{N}$ such that $i < j < N$ and $g^i = g^j$. Then $0 < j - i < N$ and $g^{j-i} = 1$. But this contradicts the minimality of N, so the elements $g^0, \dots, g^{N-1}$ must all be distinct. Therefore, the set $\{g^0, \dots, g^{N-1}\}$ has exactly N elements.

Next, we prove that $\langle g \rangle = \{g^0, \dots, g^{N-1}\}$. We know that $\{g^0, \dots, g^{N-1}\} \subset \langle g \rangle$ by definition. We must show that $\langle g \rangle \subset \{g^0, \dots, g^{N-1}\}$. Fix $m \in \mathbb{N}$. By the division algorithm, there exist integers q and r such that $m = qN + r$ and $0 \leq r < N$. Therefore,

$$g^m = g^{qN+r} = \left(g^N\right)^q g^r = 1^q g^r = g^r,$$

which implies $g^m = g^r \in \{g^0, \dots, g^{N-1}\}$. It follows that $\langle g \rangle = \{g^0, \dots, g^{N-1}\}$. This also implies that the size of $\langle g \rangle$ is N, as desired.

Finally, we prove that $\langle g \rangle$ is a subgroup of G. Note that $1 = g^0 \in \langle g \rangle$. To prove that $\langle g \rangle$ is closed under the group operation, observe that $g^m g^n = g^{m+n} \in \langle g \rangle$ for all $m, n \in \mathbb{N}$. Now, we prove that $\langle g \rangle$ is closed under inverses. Because $\langle g \rangle = \{g^0, \dots, g^{N-1}\}$, it suffices to prove that for all $i \in \mathbb{N}$ with $i < N$, the inverse of g^i is a member of $\langle g \rangle$. Since both $g^i g^{N-i} = g^N = 1$ and $g^{N-i} g^i = g^N = 1$, it follows that g^{N-i} is the inverse of g^i, and since $i < N$, we know that $g^{N-i} \in \langle g \rangle$. Thus, by Theorem 33.2, the set $\langle g \rangle$ must be a subgroup of G. □

Remark: We have defined the subgroup $\langle g \rangle$ generated by a single element $g \in G$. We can also define the subgroup $\langle S \rangle$ generated by an arbitrary subset $S \subset G$. Furthermore, although we have defined $\langle g \rangle$ only for finite groups, we can also define $\langle g \rangle$ and $\langle S \rangle$ for infinite groups. For these generalizations, consult a textbook on abstract algebra.

Definition 33.7. Let G be a finite group and let $g \in G$. The **order** of g is defined as the order of the subgroup $\langle g \rangle$ generated by g, or equivalently, as the smallest positive integer N such that $g^N = 1$. The order of g is denoted by $\text{ord}(g)$.

The order of a subgroup of G must divide the order of G.

Theorem 33.8 (Lagrange's theorem)**.** *Let G be a finite group. If H is a subgroup of G, then $|H|$ divides $|G|$. In particular,* $\text{ord}(g)$ *divides $|G|$ for all $g \in G$.*

Example 33.9. The order of any subgroup of $(\mathbb{Z}_{15}, +, 0)$ must divide $|\mathbb{Z}_{15}| = 15$, so the set $\{0, 4, 5, 8, 11, 13\}$ cannot be a subgroup of $(\mathbb{Z}_{15}, +, 0)$. The order of any subgroup of $(\mathbb{Z}_{11}^*, \cdot, 1)$ must divide $|\mathbb{Z}_{11}^*| = 10$, so the set $\{1, 2, 5, 9\}$ cannot be a subgroup of $(\mathbb{Z}_{11}^*, \cdot, 1)$.

Remark: The notation $\langle g \rangle$ is ambiguous when g belongs to more than one group, because it is not clear which group and which operation is meant. For example, consider an element $a \in \mathbb{Z}_n^*$. The subgroup $\langle a \rangle$ of $(\mathbb{Z}_n^*, \cdot, 1)$ is different from the subgroup $\langle a \rangle$ of $(\mathbb{Z}_n, +, 0)$. The notation $\text{ord}(g)$ is similarly ambiguous. Context must be used to determine which group is meant.

Example 33.10. We will compare the subgroups of $(\mathbb{Z}_7^*, \cdot, 1)$ generated by the elements of $\mathbb{Z}_7^*$ with the subgroups of $(\mathbb{Z}_7, +, 0)$ generated by the elements of $\mathbb{Z}_7$.

- The subgroups of $(\mathbb{Z}_7^*, \cdot, 1)$ generated by the elements of $\mathbb{Z}_7^*$ are shown below:

$$\begin{aligned}
\langle 1 \rangle &= \{1\}, & \text{ord}(1) &= 1,\\
\langle 2 \rangle &= \{1, 2, 4\}, & \text{ord}(2) &= 3,\\
\langle 3 \rangle &= \{1, 3, 2, 6, 4, 5\}, & \text{ord}(3) &= 6,\\
\langle 4 \rangle &= \{1, 4, 2\}, & \text{ord}(4) &= 3,\\
\langle 5 \rangle &= \{1, 5, 4, 6, 2, 3\}, & \text{ord}(5) &= 6,\\
\langle 6 \rangle &= \{1, 6\}, & \text{ord}(6) &= 2.
\end{aligned}$$

  Notice that the order of each element of $\mathbb{Z}_7^*$ divides $|\mathbb{Z}_7^*| = 6$.

- The subgroups of $(\mathbb{Z}_7, +, 0)$ generated by the elements of $\mathbb{Z}_7$ are shown below:

$$\begin{aligned}
\langle 0\rangle &= \{0\}, & \operatorname{ord}(0) &= 1,\\
\langle 1\rangle &= \{0,1,2,3,4,5,6\}, & \operatorname{ord}(1) &= 7,\\
\langle 2\rangle &= \{0,2,4,6,1,3,5\}, & \operatorname{ord}(2) &= 7,\\
\langle 3\rangle &= \{0,3,6,2,5,1,4\}, & \operatorname{ord}(3) &= 7,\\
\langle 4\rangle &= \{0,4,1,5,2,6,3\}, & \operatorname{ord}(4) &= 7,\\
\langle 5\rangle &= \{0,5,3,1,6,4,2\}, & \operatorname{ord}(5) &= 7,\\
\langle 6\rangle &= \{0,6,5,4,3,2,1\}, & \operatorname{ord}(6) &= 7.
\end{aligned}$$

  Notice that the order of each element of $\mathbb{Z}_7$ divides $|\mathbb{Z}_7| = 7$.

This example shows that the subgroups $\langle 1\rangle$, $\langle 2\rangle$, $\langle 3\rangle$, $\langle 4\rangle$, $\langle 5\rangle$, $\langle 6\rangle$ of $(\mathbb{Z}_7^*, \cdot, 1)$ are different from the subgroups $\langle 1\rangle$, $\langle 2\rangle$, $\langle 3\rangle$, $\langle 4\rangle$, $\langle 5\rangle$, $\langle 6\rangle$ of $(\mathbb{Z}_7, +, 0)$.

The following theorem generalizes Fermat's little theorem.

Theorem 33.11. *Let G be a finite group and let $g \in G$. Let $m, n \in \mathbb{Z}$. Then:*

1. $g^{\operatorname{ord}(g)} = g^{|G|} = 1$.
2. $g^n = g^{n \bmod \operatorname{ord}(g)} = g^{n \bmod |G|}$.
3. $g^m = g^n$ *if and only if* $m \equiv n \pmod{\operatorname{ord}(g)}$.

Suppose G is a finite group and $g \in G$. Let us write $x \xrightarrow{g} y$ to denote $xg = y$. The subgroup $\langle g\rangle$ generated by g has the following cyclic structure:

$$1 \xrightarrow{g} g \xrightarrow{g} g^2 \xrightarrow{g} g^3 \xrightarrow{g} \cdots \xrightarrow{g} g^{\operatorname{ord}(g)-1} \xrightarrow{g} 1$$

This motivates the following definition.

Definition 33.12. Let G be a finite group. We say that G is **cyclic** if $G = \langle g\rangle$ for some $g \in G$, in which case we also say that g **generates** G and that g is a **generator** of G.

Example 33.13. We see from Example 33.10 that the generators of $(\mathbb{Z}_7^*, \cdot, 1)$ are 3 and 5. We also see that every of element of $\mathbb{Z}_7$ except for 0 is a generator of $(\mathbb{Z}_7, +, 0)$. This last statement also follows from Lagrange's theorem and the fact that 7 is prime. Every subgroup of $(\mathbb{Z}_7, +, 0)$ must have order 1 or 7. For each $a \in \mathbb{Z}_7 \setminus \{0\}$, the subgroup $\langle a\rangle$ contains at least two elements, namely, 0 and a. Therefore, the order of $\langle a\rangle$ must be 7 (not 1), which implies $\langle a\rangle = \mathbb{Z}_7$.

Theorem 33.14. *Every finite cyclic group is commutative.*

Proof. This theorem is merely a consequence of the commutativity of integer addition and the rules for repeated application of the group operation. Let G be a finite group and g be a generator of G. Let $a, b \in G$. There exist $m, n \in \mathbb{N}$ such that $a = g^m$ and $b = g^n$. Therefore,

$$ab = g^m g^n = g^{m+n} = g^{n+m} = g^n g^m = ba.$$

□

Remark: The above theorem is also true for infinite cyclic groups, which we have not defined.

When p is prime, the group $\mathbb{Z}_p^*$ is cyclic, or in other words, is generated by a single element. For a proof of this fact, consult a textbook on abstract algebra.

Theorem 33.15. *For all primes p, the group $(\mathbb{Z}_p^*, \cdot, 1)$ is cyclic.*

34 Introduction

These notes are intended to provide a brief survey of some basic concepts in number theory. Many claims made in these notes are stated without proof. Consult your cryptography textbook or a textbook on number theory for reference.

We shall denote the set of integers by $\mathbb{Z}$, the set of natural numbers (non-negative integers) by $\mathbb{N}$, and the set of positive integers by $\mathbb{Z}_+$; that is,

$$\begin{aligned}\mathbb{Z} &= \{\dots, -3, -2, -1, 0, 1, 2, 3, \dots\},\\ \mathbb{N} &= \{0, 1, 2, 3, \dots\},\\ \mathbb{Z}_+ &= \{1, 2, 3, \dots\}.\end{aligned}$$

The above notation is not universally agreed upon: some authors use different conventions for $\mathbb{N}$ and $\mathbb{Z}_+$. The use of the symbol $\mathbb{Z}$ to denote the set of integers comes from the German word *Zahlen* for "numbers".

35 Division and remainders

We begin with a theorem on division with remainders. The theorem is commonly called the "division algorithm", even though it is a theorem, not an algorithm. The name comes from the division algorithm you learned in grade school.

Theorem 35.1 (division algorithm)**.** *For all $a \in \mathbb{Z}$ and $b \in \mathbb{Z}_+$, there exist unique integers q and r such that*

$$a = qb + r \quad \textit{and} \quad 0 \le r < b. \tag{35.1}$$

Remark: Students sometimes think the above theorem holds only when $a > b$. In fact, it also holds when $a \le b$ and even when a is negative. Although we have restricted b to the positive integers in the theorem above, there is an analogous theorem for the case when b is a negative. But we shall not need that case.

Example 35.2. The following table illustrates Theorem 35.1 with some specific numbers:

a	b	q	r
11	4	2	3
8	4	2	0
5	7	0	5
-3	4	-1	1
-9	7	-2	5

Notice, from the example above, that $q = \lfloor a/b \rfloor$ and r is the remainder obtained upon dividing a by b. Here the meaning of "remainder" must be extended to the case when a is negative.

Definition 35.3. Let $a \in \mathbb{Z}$ and $b \in \mathbb{Z}_+$. We denote the unique integer r satisfying (35.1) by $a \bmod b$.

We can then rewrite (35.1) as follows:

$$a = b\left\lfloor \frac{a}{b} \right\rfloor + a \bmod b.$$

Remark: There is some disagreement about the correct definition of $a \bmod b$ when a or b is negative. You may find that different programming language implementations define it differently. The definition given above ensures that $a \bmod b$ is an integer between 0 and $b - 1$ whenever $b > 0$.

Remark: We can compute $a \bmod b$ using the long division algorithm. Computing $a \bmod b$ is referred to as **reducing a modulo b**.

Example 35.4. Some examples of reducing one integer by another are shown below.

$$\begin{aligned}(3 + 3) \bmod 4 &= 6 \bmod 4 = 2\\ (1 - 5) \bmod 7 &= -4 \bmod 7 = 3\\ 3 \cdot 11 \bmod 12 &= 33 \bmod 12 = 9\end{aligned}$$

Efficient computation

Let a and b be k-bit integers, and let n be a k-bit positive integer. By using long division, we can compute $a \bmod n$ in space $O(k)$ and time $O(k^2)$. Similarly, by using grade-school arithmetic algorithms, we can compute $(a+b) \bmod n$ and $ab \bmod n$ in space $O(k)$ and time $O(k^2)$. Now, let $a_1, \ldots, a_m$ be k-bit integers. How much time and space does it take to compute $(a_1 + \cdots + a_m) \bmod n$ and $a_1 \cdots a_m \bmod n$? The naïve method is to compute $a_1 + \cdots + a_m$ and $a_1 \cdots a_m$ and then reduce the sum and product, respectively, modulo n. This naïve method computes $(a_1 + \cdots + a_m) \bmod n$ in space $O(k+m)$ and time $O\big((k+m)^2\big)$ and computes $a_1 \cdots a_m \bmod n$ in space $O(km)$ and time $O(k^2m^2)$. We can improve the space and time complexity by reducing modulo n after each addition or multiplication, as shown below.

---

function modsum$(n, a_1, \ldots, a_m)$
  $s := 0$ **for** $i := 1$ **to** m **do**
    $s := (s + a_i) \bmod n$
  return s
function modprod$(n, a_1, \ldots, a_m)$
  $P := 1$ **for** $i := 1$ **to** m **do**
    $P := Pa_i \bmod n$
  return P

The functions modsum and modprod both use space $O(k)$ and run in time $O(k^2m)$. The correctness of modsum and modprod follows from the theorem below.

Theorem 35.5. *For all $a, b, c \in \mathbb{Z}$ and $n \in \mathbb{Z}_+$,*

1. $(a+b+c) \bmod n = \Big(\big((a+b) \bmod n\big) + c\Big) \bmod n$;
2. $abc \bmod n = \big((ab \bmod n)\, c\big) \bmod n$.

If the integers $a_1, \ldots, a_m$ are all the same, then we can compute $a_1 \cdots a_m \bmod n$ faster than modprod by repeated squaring.

Algorithm: Modular exponentiation by repeated squaring

function modexp(n, a, m)
  Let $(m_N, \ldots, m_0)$ be the binary representation of m. $P := 1$ **for** $i := N$ **down to** 0 **do**
    $P := P^2 \bmod n$ **if** $m_i = 1$ **then** $P := Pa \bmod n$
  return P

The function modexp computes $a^m \bmod n$ in space $O(k)$ and time $O(k^2 \log m)$. To prove correctness of modexp, verify the following loop invariant: after each iteration of the for-loop,

$$P = a^{\sum_{j=i}^{N} 2^{j-i} m_j} \bmod n.$$

36 Divisibility

First, we formally define what it means for one integer to divide another.

Definition 36.1. Let a and b be integers. We say that a **divides** b and write $a \mid b$ if there exists an integer c such that $b = ca$, in which case we also say that b is **divisible** by a, a is a **divisor** of b, a is a **factor** of b, and b is a **multiple** of a. If a does not divide b, then we write $a \nmid b$.

Remark: Under the above definition, 0 divides 0; some authors exclude this case in their definition of divisibility. This distinction is not important for our discussion.

Example 36.2. Observe that

$$1 \mid 6, \quad 2 \mid 6, \quad 3 \mid 6, \quad 6 \mid 6, \quad 6 \mid 0, \quad (-3) \mid 6, \quad 3 \mid (-6), \quad (-3) \mid (-6)$$

but

$$6 \nmid 1, \quad 6 \nmid 2, \quad 6 \nmid 3, \quad 0 \nmid 6, \quad 6 \nmid (-3), \quad (-6) \nmid 3, \quad (-6) \nmid (-3).$$

The theorem below states some basic facts about divisibility.

Theorem 36.3. *For all $a, b, c \in \mathbb{Z}$:*

1. $a \mid b \;\Leftrightarrow\; (-a) \mid b \;\Leftrightarrow\; a \mid (-b) \;\Leftrightarrow\; (-a) \mid (-b)$.
2. $a \mid a$ *and* $a \mid 0$ *and* $1 \mid a$.
3. *If $a \neq 0$, then $0 \nmid a$.*
4. *If $a \mid b$, then $|a| \leq |b|$ or $b = 0$.*
5. *If $a \mid b$ and $b \mid a$, then $|a| = |b|$.*
6. *If $a \mid b$ and $b \mid c$, then $a \mid c$.*
7. *If $a \mid b$, then $a \mid bc$.*
8. *If $a \mid b$ and $a \mid c$, then $a \mid (b + c)$.*
9. *If $c \neq 0$, then $a \mid b$ if and only if $ca \mid cb$.*

37 Primality

The prime numbers are positive integers with only trivial divisors.

Definition 37.1. An integer $p > 1$ is said to be **prime** if the only positive divisors of p are 1 and p itself. An integer greater than 1 that is not prime is called **composite**.

Remark: The smallest prime number is 2, which is also the only even prime. By convention, 1 is neither prime nor composite.

Example 37.2. The integers 2, 3, 5, 7, and 11 are prime. The integers 4, 6, 8, 9, 10, and 12 are composite.

If a prime divides the product of two integers, then it must divide at least one of those integers.

Theorem 37.3. *If a and b are integers and p is prime, then $p \mid ab$ implies $p \mid a$ or $p \mid b$.*

Every positive integer has a factorization into primes that is unique up to the order of multiplication.

Theorem 37.4 (fundamental theorem of arithmetic)**.** *For every $n \in \mathbb{Z}_+$, there exist unique primes $p_1, \ldots, p_k$ and exponents $\alpha_1, \ldots, \alpha_k \in \mathbb{Z}_+$ such that $p_1 < \cdots < p_k$ and $n = p_1^{\alpha_1} \cdots p_k^{\alpha_k}$.*

Example 37.5. The prime factorization of 4312 is $2^3 \cdot 7^2 \cdot 11$.

Randomly generating primes

In cryptography, we often need to randomly generate large prime numbers. Suppose we wish to generate an n-bit prime p in which the high-order bit is 1, that is, a prime p such that $2^{n-1} \leq p < 2^n$. Consider the following strategy:

1. For $t := 1$ to T:
   (a) Choose an integer R between 2^{n-1} and $2^n - 1$ uniformly at random.
   (b) Test whether R is prime.
   (c) If R is prime, then return R.
2. Abort. (Loop above failed to find a prime.)

The above strategy is likely to succeed only if there are sufficiently many primes between 2^{n-1} and $2^n - 1$. This is true for large n, as a consequence of the prime number theorem.

Definition 37.6. For each $x \in \mathbb{N}$, let $\pi(x)$ denote the number of primes less than or equal to x.

Theorem 37.7 (prime number theorem)**.** *As $x \to \infty$, the number of primes less than or equal to x is asymptotically $x/\ln(x)$, or in other words,*

$$\lim_{x\to\infty} \frac{\pi(x)}{x/\ln(x)} = 1.$$

The prime number theorem implies that the number of primes between 2^{n-1} and $2^n - 1$ is $\Omega(2^n/n)$. Thus, if we choose an integer R between 2^{n-1} and $2^n - 1$ uniformly at random, then the probability that R is prime is at least $\Omega(1/n)$. Therefore, after $T = \Theta(n^2)$ attempts, the probability that the strategy given above fails to find a prime is at most

$$\big(1 - \Omega(1/n)\big)^T \leq e^{-\Omega(T/n)} = e^{-\Omega(n)}.$$

The inequality above follows from the fact that $1 - z \leq e^{-z}$ for every real number z.

We have not discussed how to test whether an integer is prime. We shall not say much on this topic: we only note that it can be done in polynomial time. An algorithm for determining whether an integer $N > 1$ is prime runs polynomial time if and only if it runs in time polynomial in $\log N$, that is, polynomial in the number of bits needed to represent N. An algorithm that runs in time polynomial in N would be considered too slow. One of the most commonly used primality tests is a randomized algorithm called the Miller-Rabin primality test. Given an integer $N > 1$, the Miller-Rabin test outputs either PRIME or COMPOSITE, and with high probability, the answer it outputs is correct. This probability is not taken over the choice of N, but rather over the "random coins" of the algorithm. In other words, the algorithm is given a string r of uniformly random bits as an auxiliary input, and the probability is taken over the random choice of r for a fixed input N. In 2002, Manindra Agrawal, Neeraj Kayal, and Nitin Saxena published a deterministic polynomial-time algorithm for primality testing. Even though this result is considered a great breakthrough, probabilistic primality-testing algorithms such as Miller-Rabin still run faster in practice.

Some cryptographic applications require primes of a certain form, such as strong primes. We define strong primes below, but shall not discuss how to generate them.

Definition 37.8. A prime number p is called a **strong prime** if $p = 2q + 1$, where q is also prime.

38 Greatest common divisors

In this section, we define the greatest common divisor of two integers and discuss some of its properties.

Definition 38.1. A **common divisor** of two integers a and b is an integer c such that $c \mid a$ and $c \mid b$. Now, let a and b be integers, not both zero. The **greatest common divisor** (GCD) of a and b is the largest integer d such that $d \mid a$ and $d \mid b$. The greatest common divisor of a and b is denoted by $\gcd(a, b)$. We say that a and b are **relatively prime** or **coprime** if $\gcd(a, b) = 1$.

Example 38.2. The GCDs of some specific integers are shown below:

- $\gcd(24, 36) = \gcd(36, 24) = \gcd(-24, 36) = \gcd(24, -36) = \gcd(-24, -36) = 12$
- $\gcd(24, 24) = 24$, $\gcd(24, 0) = 24$, $\gcd(24, 1) = 1$

We illustrate the concept of relatively primality with a few examples:

- 4 and 15 are relatively prime.
- 1 and 12 are relatively prime.
- 9 and 15 are not relatively prime.
- 0 and 12 are not relatively prime.

The next theorem states some basic facts about GCDs.

Theorem 38.3. *Let $a, b, c \in \mathbb{Z}$ such that a and b are not both zero. Then:*

1. $\gcd(a, b) = \gcd(b, a) = \gcd(-a, b) = \gcd(a, -b) = \gcd(-a, -b)$.

2. $\gcd(a, b) \geq 1$.
3. $\gcd(a, 1) = 1$.
4. *If* $a \neq 0$, *then* $\gcd(a, a) = \gcd(a, 0) = |a|$.
5. *If* $a \mid b$, *then* $\gcd(a, b) = |a|$.
6. *If* $c > 0$, *then* $\gcd(ca, cb) = c \gcd(a, b)$.
7. *If* $a \mid bc$ *and* $\gcd(a, b) = 1$, *then* $a \mid c$.
8. *If* $c \neq 0$ *and* $\gcd(a, c) = \gcd(b, c) = 1$, *then* $\gcd(ab, c) = 1$.
9. *If* p *is a prime, then* $\gcd(a, p) = 1$ *if and only if* $p \nmid a$.

An alternative characterization of the GCD is the following.

Theorem 38.4. *Let* a *and* b *be integers, not both zero, and let* $d \in \mathbb{Z}_+$. *Then* $d = \gcd(a, b)$ *if and only if* d *satisfies both conditions below:*

1. $d \mid a$ *and* $d \mid b$.
2. *For all* $c \in \mathbb{Z}$, *if* $c \mid a$ *and* $c \mid b$, *then* $c \mid d$.

The GCD of a and b can also be characterized in terms of integer linear combinations of a and b.

Theorem 38.5. *Let* a *and* b *be integers, at least one of which is not zero. Then there exist* $x, y \in \mathbb{Z}$ *such that* $ax + by = \gcd(a, b)$. *Moreover,* $\gcd(a, b)$ *is the smallest positive integer that can be written as a linear combination* $ax + by$ *for some* $x, y \in \mathbb{Z}$.

Proof. Define the set S as follows:

$$S := \{ax + by : x, y \in \mathbb{Z}\} \cap \mathbb{Z}_+.$$

Note that the set S is non-empty, so S has a minimum by the well-ordering principle. Let $d := \min S$. We must show that $d = \gcd(a, b)$. Since $d \in S$, there exist $x, y \in \mathbb{Z}$ such that $d = ax + by$. Therefore, since $\gcd(a, b)$ divides both a and b, it follows that $\gcd(a, b)$ also divides $ax + by = d$. This implies $\gcd(a, b) \leq d$. To complete the proof, we must show that $d \leq \gcd(a, b)$. This inequality follows if d is common divisor of a and b. We shall prove that d divides a; the proof that d divides b is analogous. By the division algorithm, there exist integers q and r such that $a = qd + r$ and $0 \leq r < d$. Since $d = ax + by$, it follows that

$$a = q(ax + by) + r,$$

or equivalently,

$$r = a(1 - qx) - qby.$$

Now, combine the fact that $0 \leq r < d$ with the above equality. If $r \neq 0$, then r would be a member of S smaller than d, contradicting the minimality of d. Therefore, $r = 0$ and hence $a = qd + r = qd$, which shows that d divides a. □

Remark: If a pair of integers (x_0, y_0) satisfies $ax_0 + by_0 = \gcd(a, b)$, then

$$a\left(x_0 + \frac{kb}{\gcd(a, b)}\right) + b\left(y_0 - \frac{ka}{\gcd(a, b)}\right) = \gcd(a, b) \quad \text{for all } k \in \mathbb{Z},$$

so there are infinitely many pairs of integers (x, y) satisfying $ax + by = \gcd(a, b)$.

Least common multiples are normally discussed along with greatest common divisors.

Definition 38.6. Let a and b be integers, both not zero. The **least common multiple** (LCM) of a and b is the smallest $m \in \mathbb{Z}_+$ such that $a \mid m$ and $b \mid m$. The least common multiple of a and b is written $\operatorname{lcm}(a, b)$.

The relationship between the GCD and LCM is as follows.

Theorem 38.7. *If* a *and* b *are integers, both not zero, then* $\gcd(a, b) \operatorname{lcm}(a, b) = |ab|$.

The Euclidean and extended Euclidean algorithms

The Euclidean algorithm is a simple recursive algorithm for computing the GCD of two integers.

Algorithm: Euclidean algorithm

Input: integers a and b, not both zero, such that $a \geq b \geq 0$
Output: the greatest common divisor of a and b

function $\mathsf{EuclidGCD}(a, b)$
  if $b = 0$ **then return** a **else return** $\mathsf{EuclidGCD}(b, a \bmod b)$

Remark: We have restricted the above algorithm to the case $a \geq b \geq 0$. To compute the GCD in the other cases, negate and swap the arguments to $\mathsf{EuclidGCD}$ as appropriate. Note that the invariant on the parameters is maintained during recursive calls because $b > a \bmod b \geq 0$.

To prove the correctness of the Euclidean algorithm, we require the following lemma.

Lemma 38.8. *For all $a \in \mathbb{Z}$ and $b \in \mathbb{Z}_+$,*

$$\gcd(a, b) = \gcd(b, a \bmod b).$$

Proof. Recall that

$$a = b\left\lfloor \frac{a}{b} \right\rfloor + a \bmod b. \tag{38.1}$$

The above equality is merely a form of the division algorithm. We may rewrite (38.1) as

$$a \bmod b = a - b\left\lfloor \frac{a}{b} \right\rfloor. \tag{38.2}$$

It follows from (38.1) that every common divisor of b and $(a \bmod b)$ also divides both a and b. It follows from (38.2) that every common divisor of a and b also divides both b and $(a \bmod b)$. Therefore, the common divisors of a and b are exactly the same as the common divisors of b and $(a \bmod b)$. This implies that $\gcd(a, b) = \gcd(b, a \bmod b)$. □

Theorem 38.9. *Let a and b be integers, not both zero, such that $a \geq b \geq 0$. Assume that a and b are both k bits long. Then $\mathsf{EuclidGCD}(a, b)$ returns $\gcd(a, b)$ and runs in time $O(k^3)$.*

Proof. We prove the correctness of $\mathsf{EuclidGCD}$ by induction on b. First, consider the base case $b = 0$:

$$\mathsf{EuclidGCD}(a, 0) = a = \gcd(a, 0).$$

Now, suppose $b > 0$ and assume the following induction hypothesis: $\mathsf{EuclidGCD}(a', b') = \gcd(a', b')$ for all integers a' and b', not both zero, such that $a' \geq b' \geq 0$ and $b' < b$. Note that $b > a \bmod b \geq 0$. Thus, by the induction hypothesis,

$$\mathsf{EuclidGCD}(a, b) = \mathsf{EuclidGCD}(b, a \bmod b) = \gcd(b, a \bmod b).$$

Since $\gcd(b, a \bmod b) = \gcd(a, b)$ by Lemma 38.8, it follows that $\mathsf{EuclidGCD}(a, b) = \gcd(a, b)$.

Now, we bound the running time of $\mathsf{EuclidGCD}$. The integer operations performed during each recursive call to $\mathsf{EuclidGCD}$ take $O(k^2)$ time. Thus, it suffices to show that the number of recursive calls is $O(k)$. We shall prove that the second parameter decreases by at least half after at most two recursive calls, so the number of recursive calls is $O(\log b) = O(k)$. Notice that since $a \bmod b < b$, the second parameter decreases after each call. Let $r := a \bmod b$. If $r \leq b/2$, then the second parameter decreases by at least half after one recursive call. Now, suppose $r > b/2$. After one recursive call, the second parameter is r; after another recursive call, the second parameter is $b \bmod r$. We must prove that $b \bmod r \leq b/2$. Recall that

$$b = r\left\lfloor \frac{b}{r} \right\rfloor + b \bmod r. \tag{38.3}$$

Since $r = a \bmod b < b$, it follows that $b/r > 1$ and hence $\lfloor b/r \rfloor \geq 1$. (In fact, $\lfloor b/r \rfloor = 1$, but we shall not need exact equality in the proof.) Since $\lfloor b/r \rfloor \geq 1$ and $r > b/2$, it follows from (38.3) that

$$b \bmod r = b - r\left\lfloor \frac{b}{r} \right\rfloor \leq b - r < \frac{b}{2}.$$

□

Example 38.10. Suppose we wish to compute the greatest common divisor of 693 and 153. We can execute the Euclidean algorithm by hand as follows:

$$\gcd(693, 153) = \gcd(153, 81) = \gcd(81, 72) = \gcd(72, 9) = \gcd(9, 0) = 9.$$

Recall that Theorem 38.5 states that there exist integers x and y such that $ax + by = \gcd(a, b)$. The Euclidean algorithm can be extended to find such integers x and y.

Algorithm: Extended Euclidean algorithm

Input: integers a and b, not both zero, such that $a \geq b \geq 0$
Output: $(d, x, y) \in \mathbb{Z}^3$ such that $ax + by = d = \gcd(a, b)$

function ExtGCD(a, b)
  if $b = 0$ **then return** $(a, 1, 0)$ **else**
    $q := \lfloor a/b \rfloor$, $r := a \bmod b$ $(d, X, Y) :=$ ExtGCD(b, r) **return** $(d, Y, X - qY)$

Remark: We have restricted the above algorithm to the case $a \geq b \geq 0$. For the other cases, negate and swap the inputs and outputs of ExtGCD as appropriate. Note that the invariant on the parameters is maintained during recursive calls because $b > r = a \bmod b \geq 0$.

Theorem 38.11. *Let a and b be integers, not both zero, such that $a \geq b \geq 0$. Then* ExtGCD(a, b) *outputs a triple of integers (d, x, y) such that $ax + by = d = \gcd(a, b)$. Furthermore, if a and b are both k bits long, then* ExtGCD(a, b) *runs in time $O(k^3)$.*

Proof. Let (d, x, y) denote the output of ExtGCD(a, b). To prove that $d = \gcd(a, b)$, we just follow the same steps used in Theorem 38.9 to prove the correctness of EuclidGCD. Thus, to prove the correctness of ExtGCD, it suffices to prove $ax + by = d$. We prove this by induction on b. First, consider the base case $b = 0$:

$$ax + by = a \cdot 1 + 0 \cdot 0 = a = d.$$

Now, suppose $b > 0$ and assume the following induction hypothesis: ExtGCD(a', b') returns a triple of integers (d', x', y') such that $a'x' + b'y' = d'$ for all integers a' and b', not both zero, such that $a' \geq b' \geq 0$ and $b' < b$. Let q, r, X, and Y be as shown in the definition of ExtGCD. Since $b > r = a \bmod b \geq 0$, it follows from the induction hypothesis that $bX + rY = d$. We know from the division algorithm that $a = qb + r$. Therefore,

$$ax + by = (qb + r)Y + b(X - qY) = bX + rY = d.$$

Now, we assume that a and b are both k bits long and bound the running time of ExtGCD(a, b). The number of recursive calls made by ExtGCD is $O(\log b) = O(k)$, the same as the number recursive calls made by EuclidGCD. To prove this, we just follow the same steps as in Theorem 38.9. Although ExtGCD performs more integer operations per recursive call than EuclidGCD, these operations still take $O(k^2)$ time per call. Thus, the total running time of ExtGCD is $O(k^3)$. □

Example 38.12. Suppose we wish to find integers x and y such that $693x + 153y = \gcd(693, 153)$. We can execute the extended Euclidean algorithm by hand as follows. First, we compute the GCD of 693 and 153, saving the quotients and remainders at each step:

$$\begin{aligned} 693 &= 4 \cdot 153 + 81 \\ 153 &= 1 \cdot 81 + 72 \\ 81 &= 1 \cdot 72 + 9 \\ 72 &= 8 \cdot 9 + 0 \end{aligned}$$

Thus, the GCD of 693 and 153 is 9. Now, we reverse the previous steps to find x and y:

$$\begin{aligned} 9 &= 81 - 72 \\ &= 81 - (153 - 81) \\ &= -153 + 2 \cdot 81 \\ &= -153 + 2 \cdot (693 - 4 \cdot 153) \\ &= 2 \cdot 693 - 9 \cdot 153 \end{aligned}$$

Thus, we obtain $x = 2$ and $y = -9$.

39 Congruence

In this section, we introduce the concepts of congruence and arithmetic modulo a positive integer (commonly called **modular arithmetic**).

Definition 39.1. Let $a, b \in \mathbb{Z}$ and $n \in \mathbb{Z}_+$. We say that a is **congruent to** b **modulo** n if $n \mid (a - b)$, in which case we write $a \equiv b \pmod{n}$. If a is not congruent to b modulo n, then we write $a \not\equiv b \pmod{n}$. The integer n is called the **modulus**.

Remark: The definition of $a \equiv b \pmod{n}$ can be restated in the following two ways:

1. $a - b$ is a multiple of n.
2. There exists an integer k such that $a - b = kn$.

The first thing to notice about congruence modulo n is that it is an equivalence relation.

Theorem 39.2. *Congruence modulo a positive integer is an equivalence relation. In other words, for all $n \in \mathbb{Z}_+$ and $a, b, c \in \mathbb{Z}$, the following properties hold:*

reflexivity: $a \equiv a \pmod{n}$.

symmetry: *If* $a \equiv b \pmod{n}$, *then* $b \equiv a \pmod{n}$.

transitivity: *If* $a \equiv b \pmod{n}$ *and* $b \equiv c \pmod{n}$, *then* $a \equiv c \pmod{n}$.

But congruence modulo n is more than just an equivalence relation: it is also conserved under the usual operations of integer arithmetic.[2]

Theorem 39.3. *For all $n \in \mathbb{Z}_+$ and $a, b, c, d \in \mathbb{Z}$:*

1. *If* $a \equiv b \pmod{n}$ *and* $c \equiv d \pmod{n}$, *then* $a + c \equiv b + d \pmod{n}$.
2. *If* $a \equiv b \pmod{n}$ *and* $c \equiv d \pmod{n}$, *then* $ac \equiv bd \pmod{n}$.
3. *If* $a \equiv b \pmod{n}$, *then* $-a \equiv -b \pmod{n}$.
4. *If* $a \equiv b \pmod{n}$ *and* $c \equiv d \pmod{n}$, *then* $a - c \equiv b - d \pmod{n}$.

Remark on notation: Notice that the symbol "mod" has two different meanings:

reduction: $a \bmod b$ is the result of applying the mod operator to the arguments a and b.

congruence: $a \equiv b \pmod{n}$ expresses a Boolean relationship between the integers a and b.

Some authors prefer to write $a = b \pmod{n}$ or $a = b \mod n$ instead of $a \equiv b \pmod{n}$. Your cryptography textbook uses a different notation from the one used in these notes. These differences are explained in the table below.

	these notes	textbook
reduction	$a \bmod b$	$[a \bmod b]$
congruence	$a \equiv b \pmod{n}$	$a = b \mod n$

[2] Together, Theorem 39.2 and Theorem 39.3, parts 1 and 2, say that congruence modulo n is a *congruence relation*.

Example 39.4. The left column below shows some calculations with reduction, and the right column shows their corresponding calculations with congruence.

$(3+3) \bmod 4 = 6 \bmod 4 = 2$	$3+3 \equiv 6 \equiv 2 \pmod 4$
$(1-5) \bmod 7 = -4 \bmod 7 = 3$	$1-5 \equiv -4 \equiv 3 \pmod 7$
$3 \cdot 11 \bmod 12 = 33 \bmod 12 = 9$	$3 \cdot 11 \equiv 33 \equiv 9 \pmod{12}$

The example above exhibits a relationship between congruence and reduction. That relationship is stated precisely by the theorem below.

Theorem 39.5. *For all $a, b \in \mathbb{Z}$ and $n \in \mathbb{Z}_+$,*

1. *$a \equiv b \pmod n$ if and only if $a \bmod n = b \bmod n$;*
2. *$a \bmod n$ is the unique integer r such that $0 \leq r < n$ and $r \equiv a \pmod n$.*

Notice that Theorem 35.5 follows from Theorems 39.5, 39.3, and 39.2. We end with two more basic facts about congruence.

Theorem 39.6. *For all $a, b \in \mathbb{Z}$ and $m, n \in \mathbb{Z}_+$:*

1. *If $a \equiv b \pmod n$, then $ma \equiv mb \pmod{mn}$.*
2. *If $a \equiv b \pmod n$ and $m \mid n$, then $a \equiv b \pmod m$.*

39.1 Multiplicative inverses

Let a be an integer and n be a positive integer. There exists an integer b such that $a + b \equiv 0 \pmod n$; for example, take $b = -a$. Now, consider multiplication instead of addition. Does there always exist an integer b such that $ab \equiv 1 \pmod n$? In general, the answer to this question is no. Not every integer has a multiplicative inverse modulo n.

Definition 39.7. Let $a \in \mathbb{Z}$ and $n \in \mathbb{Z}_+$. A **multiplicative inverse** of a **modulo** n is an integer b such that $ab \equiv 1 \pmod n$.

Example 39.8. No integer between 0 and 5 is a multiplicative inverse of 2 modulo 6, because

$$2 \cdot 0 \equiv 0 \pmod 6, \quad 2 \cdot 1 \equiv 2 \pmod 6, \quad 2 \cdot 2 \equiv 4 \pmod 6,$$
$$2 \cdot 3 \equiv 0 \pmod 6, \quad 2 \cdot 4 \equiv 2 \pmod 6, \quad 2 \cdot 5 \equiv 4 \pmod 6.$$

Now, suppose that some integer b is a multiplicative inverse of 2 modulo 6. Then $b \bmod 6$ would also be a multiplicative inverse of 2 modulo 6, because $2(b \bmod 6) \equiv 2b \equiv 1 \pmod 6$. But $0 \leq b \bmod 6 \leq 5$ and we showed above that no integer between 0 and 5 is a multiplicative inverse of 2 modulo 6, so we have a contradiction. Hence, no integer is a multiplicative inverse of 2 modulo 6.

An integer with a multiplicative inverse modulo n can be canceled if it appears as a factor on both sides of a congruence modulo n.

Theorem 39.9 (cancellation)**.** *Let $n \in \mathbb{Z}_+$ and $c, x, y \in \mathbb{Z}$. If c has a multiplicative inverse modulo n, then $cx \equiv cy \pmod n$ implies $x \equiv y \pmod n$.*

Proof. Suppose c has a multiplicative inverse b modulo n. If $cx \equiv cy \pmod n$, then

$$x \equiv 1 \cdot x \equiv (cb)x \equiv (bc)x \equiv b(cx) \equiv b(cy) \equiv (bc)y \equiv (cb)y \equiv 1 \cdot y \equiv y \pmod n.$$

□

Suppose b is a multiplicative inverse of a modulo n. Then every integer congruent to b modulo n is also a multiplicative inverse of a modulo n, so a has infinitely many multiplicative inverses modulo n. However, only one integer between 0 and $n-1$ can be a multiplicative inverse of a modulo n.

Theorem 39.10. *Let $a \in \mathbb{Z}$ and $n \in \mathbb{Z}_+$. Then:*

1. *Any two multiplicative inverses of a modulo n are congruent to each other modulo n.*
2. *If a has at least one multiplicative inverse modulo n, then a has exactly one multiplicative inverse modulo n between 0 and $n-1$ (inclusive).*

Proof. First, we prove part 1. Suppose b and b' are both multiplicative inverses of a modulo n. By definition, $ab \equiv 1 \equiv ab' \pmod{n}$, and hence $b \equiv b' \pmod{n}$ by cancellation.

Now, we prove part 2. Suppose that a has some multiplicative inverse x modulo n. Then $x \bmod n$ is multiplicative inverse of a modulo n between 0 and $n-1$, because $a(x \bmod n) \equiv ax \equiv 1 \pmod{n}$ and $0 \leq x \bmod n < n$. Now, let y be a multiplicative inverse of a modulo n such that $0 \leq y < n$. We must show that $y = x \bmod n$. It follows from part 1 that $x \equiv y \pmod{n}$; hence, $x \bmod n = y \bmod n$. Since $0 \leq y < n$, we know that $y = y \bmod n$. Therefore, $y = y \bmod n = x \bmod n$. □

If two integers both have multiplicative inverses modulo n, their product also has a multiplicative inverse modulo n. In other words, the existence of multiplicative inverses modulo n is closed under multiplication.

Theorem 39.11. *Let $a, b \in \mathbb{Z}$ and $n \in \mathbb{Z}_+$. If a' and b' are respectively multiplicative inverses of a and b modulo n, then $b'a'$ is a multiplicative inverse of ab modulo n.*

Proof. If $aa' \equiv 1 \pmod{n}$ and $bb' \equiv 1 \pmod{n}$, then

$$(ab)(b'a') \equiv \big((ab)b'\big)a' \equiv \big(a(bb')\big)a' \equiv (a \cdot 1)a' \equiv aa' \equiv 1 \pmod{n}.$$

□

Now, we consider the question of which integers have multiplicative inverses modulo n. We prove next that the integers with a multiplicative inverse modulo n are precisely those that are relatively prime to n.

Theorem 39.12. *Let $a \in \mathbb{Z}$ and $n \in \mathbb{Z}_+$. Then some integer is a multiplicative inverse of a modulo n if and only if $\gcd(a, n) = 1$.*

Proof. Suppose a has some multiplicative inverse b modulo n. Then $ab \equiv 1 \pmod{n}$, or equivalently, there exists an integer k such that $ab - 1 = kn$. Since $\gcd(a, n)$ divides both a and n, it follows that $\gcd(a, n)$ also divides $ab - kn = 1$. This implies $\gcd(a, n) = 1$.

Now, we prove the converse. Assume that $\gcd(a, n) = 1$. By Theorem 38.5, there exist integers x and y such that $ax + ny = \gcd(a, n) = 1$. Therefore, $ax \equiv ax + 0 \equiv ax + ny \equiv 1 \pmod{n}$. Hence, we conclude that x is a multiplicative inverse of a modulo n. □

Corollary 39.13. *Let $a \in \mathbb{Z}$ and p be prime. Then some integer is a multiplicative inverse of a modulo p if and only if $p \nmid a$. In particular, the integers $1, \ldots, p-1$ all have multiplicative inverses modulo p.*

Proof. By Theorem 39.12, we know that a has a multiplicative inverse modulo p if and only if $\gcd(a, p) = 1$. To complete the proof, observe that $\gcd(a, p) = 1$ if and only if $p \nmid a$ and that none of the integers $1, \ldots, p-1$ is divisible by p. □

Example 39.14. We showed in Example 39.8 that 2 has no multiplicative inverses modulo 6 by eliminating all possibilities. Theorem 39.12 provides an easier way to show this: just check whether $\gcd(2, 6)$ equals 1. Since $\gcd(2, 6) = 2$, it follows that 2 does not have a multiplicative inverse modulo 6.

Computing multiplicative inverses

The proof of Theorem 39.12 suggests a method for computing multiplicative inverses modulo n. Suppose we wish to find a multiplicative inverse of a modulo n. We can use the extended Euclidean algorithm to find integers d, x, and y such that $ax + ny = d = \gcd(a, b)$. If $d \neq 1$, then we know that a has no multiplicative inverse modulo n. If $d = 1$, then $ax \equiv ax + ny \equiv 1 \pmod{n}$, so x is a multiplicative inverse of a modulo n. In that case, $x \bmod n$ is the multiplicative inverse of a modulo n between 0 and $n-1$.

Example 39.15. Suppose we wish to find a multiplicative inverse of 7 modulo 11. We can execute the extended Euclidean algorithm by hand as follows. First, we compute the GCD of 7 and 11, saving the quotients and remainders at each step:

$$\begin{aligned} 11 &= 1 \cdot 7 + 4 \\ 7 &= 1 \cdot 4 + 3 \\ 4 &= 1 \cdot 3 + 1 \\ 4 &= 1 \cdot 3 + 1 \\ 3 &= 3 \cdot 1 + 0 \end{aligned}$$

Thus, the GCD of 7 and 11 is 1, so we know that 7 must have a multiplicative inverse modulo 11. Now, we reverse the previous steps to find integers x and y such that $7x + 11y = 1$.

$$\begin{aligned} 1 &= 4 - 3 \\ &= 4 - (7 - 4) \\ &= -7 + 2 \cdot 4 \\ &= -7 + 2 \cdot (11 - 7) \\ &= -3 \cdot 7 + 2 \cdot 11 \end{aligned}$$

Thus, we obtain $x = -3$ and $y = 2$. Reducing -3 modulo 11 gives 8, which is the multiplicative inverse of 7 modulo 11 that lies between 0 and 10.

39.2 Exponentiation

We saw in Section 35 how to efficiently perform modular exponentiation by repeated squaring. When the modulus is a prime p, we can further speed up the calculation by first reducing the exponent modulo $p - 1$. This is a consequence of Fermat's little theorem, which will we prove below. First, we prove the following two lemmas.

Lemma 39.16. *Let p be a prime. Then, for $1 \leq k \leq p - 1$, the binomial coefficient $\binom{p}{k}$ is a multiple of p.*

Proof. Since the binomial coefficient $\binom{p}{k}$ is an integer and

$$\binom{p}{k} = \frac{p!}{k!(p-k)!},$$

we know that $k!(p-k)!$ divides $p!$. We claim that p does not divide $k!(p-k)!$. Suppose, to the contrary, that p does divide $k!(p-k)!$. Then p would have to divide some integer in the set

$$S_k := \{1, \ldots, k\} \cup \{1, \ldots, p-k\}.$$

But for $1 \leq k \leq p-1$, every member of S_k is between 1 and $p-1$ and thus is not divisible by p. Hence, p cannot divide $k!(p-k)!$. Since p is prime, it follows that $\gcd\big(p, k!(p-k)!\big) = 1$. Since we also know that $k!(p-k)!$ divides $p! = p(p-1)!$, it follows that $k!(p-k)!$ divides $(p-1)!$. Therefore, $(p-1)! \,/\, k!(p-k)!$ is an integer, so $\binom{p}{k}$ is a multiple of p. □

Lemma 39.17. *If p is a prime, then $a^p \equiv a \pmod{n}$ for all $a \in \mathbb{Z}$.*

Proof. First, we prove the lemma for all $a \in \mathbb{N}$ by induction. For the base case $a = 0$, we just verify that $0^p \equiv 0 \pmod{n}$. For the induction hypothesis, we assume that $a^p \equiv a \pmod{p}$. Now, we prove that $(a+1)^p \equiv a + 1 \pmod{p}$. By the binomial theorem,

$$(a+1)^p \equiv \sum_{k=0}^{p} \binom{p}{k} a^k \equiv a^p + 1 + \sum_{k=1}^{p-1} \binom{p}{k} \pmod{p}. \tag{39.1}$$

We know from Lemma 39.16 that

$$\binom{p}{k} \equiv 0 \pmod{n} \quad \text{for } 1 \leq k \leq p-1. \tag{39.2}$$

Together, congruences (39.1) and (39.2) and the induction hypothesis imply

$$(a+1)^p \equiv a^p + 1 + \sum_{k=1}^{p-1} \binom{p}{k} \equiv a^p + 1 \equiv a + 1 \pmod{p}.$$

Now, consider the case that a is a negative integer. Since $a \bmod p$ is a positive integer, it follows from the argument above that $(a \bmod p)^p \equiv (a \bmod p) \pmod{p}$. Therefore,

$$a^p \equiv (a \bmod p)^p \equiv (a \bmod p) \equiv a \pmod{p}.$$

□

Theorem 39.18 (Fermat's little theorem)**.** *Let $a \in \mathbb{Z}$ and p be prime. If $p \nmid a$, then $a^{p-1} \equiv 1 \pmod{p}$.*

Proof. Suppose $p \nmid a$. By Corollary 39.13, the integer a has a multiplicative inverse modulo p. By Lemma 39.17,

$$aa^{p-1} \equiv a^p \equiv a \equiv a \cdot 1 \pmod{p}.$$

Since a has a multiplicative inverse modulo p, we can cancel a to obtain $a^{p-1} \equiv 1 \pmod{p}$. □

Corollary 39.19. *Let $a, m \in \mathbb{Z}$ and p be prime. If $p \nmid a$, then $a^m \equiv a^{m \bmod (p-1)} \pmod{p}$.*

Proof. Let $q := \lfloor m/(p-1) \rfloor$ and $r := m \bmod (p-1)$. Recall that $m = q(p-1) + r$ by the division algorithm. If $p \nmid a$, then Fermat's little theorem implies

$$a^m \equiv a^{q(p-1)+r} \equiv a^{q(p-1)} a^r \equiv \left(a^{p-1}\right)^q a^r \equiv 1^q a^r \equiv 1 \cdot a^r \equiv a^r \pmod{p}.$$

□

Remark: Suppose p is prime and $p \nmid a$. Fermat's little theorem implies that a^{p-2} must be a multiplicative inverse of a modulo p.

Example 39.20. Suppose we wish to find a multiplicative inverse of 2 modulo 7. Since 7 is prime and $7 \nmid 2$, we can apply Fermat's little theorem. We calculate $2^5 \equiv 32 \equiv 4 \pmod{7}$. It follows that 4 is a multiplicative inverse of 2 modulo 7. Another way to solve this problem is to use the extended Euclidean algorithm as shown in Example 39.15.

Example 39.21. Suppose we wish to compute $5^{8331563} \bmod 11$. Note that 11 is prime and $11 \nmid 5$, so we can apply Corollary 39.19. We calculate

$$5^{8331563} \equiv 5^{8331563 \bmod 10} \equiv 5^3 \equiv 125 \equiv 4 \pmod{11}.$$

Therefore, $5^{8331563} \bmod 11 = 4$.

40 Introduction

We saw in previous lectures how to perform certain number-theoretic operations efficiently:

- addition in $\mathbb{Z}_n$
- multiplication in $\mathbb{Z}_n$
- finding inverses in $\mathbb{Z}_n^*$
- exponentiation in $\mathbb{Z}_n$

The goal of future lectures is to build public-key cryptographic systems using these operations as primitives. To prove the security of a cryptographic system, we show that any polynomial-time attacker that breaks the system yields a polynomial-time algorithm for solving a computationally difficult problem. We shall describe several such problems below. These problems are assumed to be hard, but so far no one knows how to prove that they actually are hard. Any such proof would imply $\mathsf{P} \neq \mathsf{NP}$. Thus, the security of our cryptographic systems rests on assumptions that are widely believed, but not proven.

41 Number-theoretic assumptions

The three computational problems we describe are all relative to a specific group-generation algorithm $\mathcal{G}$, which we define below. Other choices for $\mathcal{G}$ are possible. The choice of $\mathcal{G}$ matters for security: a poor choice may make one or more of the problems described below easy.

Let p be a prime of the form $p = 2q + 1$, where q is also prime. Recall that such a prime p is called a *strong prime*. The group $\mathbb{Z}_p^*$ has a subgroup $\mathbb{G}$ of order q. The group $\mathbb{G}$ is known as the group of *quadratic residues modulo p*. Every element of $\mathbb{G}$ except for 1 is a generator of $\mathbb{G}$.

We define the algorithm $\mathcal{G}$ as follows.

---

algorithm $\mathcal{G}$:

1. Take 1^n as input (where n is the security parameter).
2. Randomly choose an $(n+1)$-bit prime p of the form $p = 2q + 1$, where q is also prime.
3. Let $\mathbb{G}$ denote the subgroup of $\mathbb{Z}_p^*$ of order q.
4. Randomly choose a generator g of $\mathbb{G}$.
5. Output (p, g).

Remark: Your textbook considers a more general case in which $\mathcal{G}$ outputs a polynomial-sized description of a cyclic group $\mathbb{G}$ of order $q \geq 2^{n-1}$, the order q of $\mathbb{G}$, and a generator g of $\mathbb{G}$. For example, suppose $\mathcal{G}$ chooses a prime p of the form $p = 2q + 1$, where q is also prime, and an element $g \in \mathbb{Z}_p^*$ of order q. Then the pair (p, g) would be a polynomial-sized description of the group $\mathbb{G}$. Since g is already included in (p, g) and since q can be derived from p, it suffices for $\mathcal{G}$ to just output (p, g).

We now describe three problems that are widely believed to be hard.

Definition 41.1. We say that an attacker $\mathcal{A}$ solves the **discrete logarithm (DL) problem** for $\mathcal{G}$ if $\mathcal{A}$ wins the following security game:

1. The challenger runs $\mathcal{G}(1^n)$ to generate (p, g), where $p = 2q + 1$.
2. The challenger chooses $x \in \mathbb{Z}_q$ uniformly at random.
3. The challenger gives (p, g, g^x) to $\mathcal{A}$.
4. The attacker $\mathcal{A}$ outputs $x' \in \mathbb{Z}_q$.
5. The attacker $\mathcal{A}$ wins the security game if and only if $x = x'$.

The **discrete logarithm (DL) assumption** for $\mathcal{G}$ is the assumption that no polynomial-time attacker can solve the discrete logarithm problem for $\mathcal{G}$ with more than negligible probability.

Definition 41.2. We say that an attacker $\mathcal{A}$ solves the **computational Diffie-Hellman (CDH) problem** for $\mathcal{G}$ if $\mathcal{A}$ wins the following security game:

1. The challenger runs $\mathcal{G}(1^n)$ to generate (p, g), where $p = 2q + 1$.
2. The challenger chooses $a, b \in \mathbb{Z}_q$ uniformly at random.
3. The challenger gives (p, g, g^a, g^b) to $\mathcal{A}$.
4. The attacker $\mathcal{A}$ wins the security game if and only if $\mathcal{A}$ outputs g^{ab}.

The **computational Diffie-Hellman (CDH) assumption** for $\mathcal{G}$ is the assumption that no polynomial-time attacker can solve the computational Diffie-Hellman problem for $\mathcal{G}$ with more than negligible probability.

Definition 41.3. We say that an attacker $\mathcal{A}$ solves the **decisional Diffie-Hellman (DDH) problem** for $\mathcal{G}$ if $\mathcal{A}$ wins the following security game:

1. The challenger runs $\mathcal{G}(1^n)$ to generate (p, g), where $p = 2q + 1$.
2. The challenger chooses $a, b, c \in \mathbb{Z}_q$ uniformly at random.
3. The challenger chooses $d \in \{0, 1\}$ uniformly at random.
4. The challenger sets
$$T := \begin{cases} g^{ab} & \text{if } d = 0, \\ g^c & \text{if } d = 1. \end{cases}$$
5. The challenger gives (p, g, g^a, g^b, T) to $\mathcal{A}$.
6. The attacker $\mathcal{A}$ outputs $d' \in \{0, 1\}$.
7. The attacker $\mathcal{A}$ wins the security game if and only if $d = d'$.
8. The advantage of $\mathcal{A}$ is defined as $\Pr[d = d'] - 1/2$.

The **decisional Diffie-Hellman (DDH) assumption** for $\mathcal{G}$ is the assumption that no polynomial-time attacker can solve the decisional Diffie-Hellman problem for $\mathcal{G}$ with more than negligible advantage.

Remark: If c uniformly distributed in $\mathbb{Z}_q$, then g^c is uniformly distributed in $\mathbb{G}$, the subgroup of $\mathbb{Z}_p^*$ of order q.

We can compare the relative difficulty of the above problems as follows:

1. If there is a polynomial-time attacker that solves the DL problem for $\mathcal{G}$ with more than negligible probability, then there is a polynomial-time attacker that solves the CDH problem for $\mathcal{G}$ with more than negligible probability.

   Informally: If the DL problem is easy, then the CDH problem is easy.
2. If there is a polynomial-time attacker that solves the CDH problem for $\mathcal{G}$ with more than negligible probability, then there is a polynomial-time attacker that solves the DDH problem for $\mathcal{G}$ with more than negligible advantage.

   Informally: If the CDH problem is easy, then the DDH problem is easy.

We can restate 1 and 2 above in the contrapositive as follows:

1′. If the CDH assumption for $\mathcal{G}$ is true, then the DL assumption for $\mathcal{G}$ is true.

   Informally: If the CDH problem is hard, then the DL problem is hard.

2′. If the DDH assumption for $\mathcal{G}$ is true, then the CDH assumption for $\mathcal{G}$ is true.

   Informally: If the DDH problem is hard, then the CDH problem is hard.

Statement 1 can be proven by giving a reduction from the CDH problem to DL problem, and statement 2 can be proven by giving a reduction from the DDH problem to the CDH problem.

42 Diffie-Hellman key exchange

The Diffie-Hellman key-exchange protocol allows two parties to securely generate a shared secret key by exchanging messages in the presence eavesdropper. Let us call the two parties in the protocol Alice and Bob. The protocol is described below.

---

1. Alice runs $\mathcal{G}(1^n)$ to generate (p, g), where $p = 2q + 1$.
2. Alice chooses $a \in \mathbb{Z}_q$ uniformly at random.
3. Alice computes g^a and sends (p, g, g^a) to Bob.
4. Bob chooses $b \in \mathbb{Z}_q$ uniformly at random.
5. Bob computes g^b and sends g^b to Alice.
6. Alice computes $(g^b)^a$ and Bob computes $(g^a)^b$.
7. The shared key of Alice and Bob is g^{ab}.

It can be shown that under DDH assumption for $\mathcal{G}$, no polynomial-time eavesdropper can distinguish the shared key of Alice and Bob from a uniformly random element in $\mathbb{G}$, the subgroup of $\mathbb{Z}_p^*$ of order q.

43 Collision-resistant hash functions

Intuitively, a collision-resistant hash function is a function for which it is hard to find two different inputs that hash to the same value. We shall give a formal definition of collision resistance below. But first, we establish some notation. Let $n \in \mathbb{N}$. Let $[n]$ denote the set of integers from 1 to n; that is, $[n] := \{1, \ldots, n\}$. Let $\{0,1\}^*$ denote the set of strings in which each character is either 0 or 1, and let $\{0,1\}^n$ denote the set of strings in $\{0,1\}^*$ of length n. Let $\{0,1\}^{\leq n}$ denote the set of strings in $\{0,1\}^*$ of length at most n and $\{0,1\}^{<n}$ denote the set of strings in $\{0,1\}^*$ of length less than n; that is,

$$\{0,1\}^{\leq n} := \bigcup_{k=0}^{n} \{0,1\}^k \quad \text{and} \quad \{0,1\}^{<n} := \bigcup_{k=0}^{n-1} \{0,1\}^k.$$

Note that $\{0,1\}^{<n} = \{0,1\}^{\leq n-1}$. Let 0^n denote the string of n zeros and 1^n denote the string of n ones. For each string x, let $|x|$ denote the length of x.[3] For all strings x and y, let $x \parallel y$ denote the concatenation of x and y.

Definition 43.1. Let $L, \ell \in \mathbb{N}$. If f is a function whose domain is either $\{0,1\}^L$ or $\{0,1\}^{\leq L}$ and whose range[4] is $\{0,1\}^\ell$, then we say that f is **compressing** if $L > \ell$. A **hash function** is a function that is compressing. A pair (x, x') of strings in the domain of f such that $x \neq x'$ and $f(x) = f(x')$ is called a **collision** for f.

Definition 43.2. A **family of functions** is an ordered pair (Gen, F) satisfying the following conditions:

- Gen is a probabilistic polynomial-time algorithm that takes 1^n as input (where n denotes the security parameter) and outputs a key k such that n can be recovered from k in polynomial time.
- There exist functions $\ell, L : \mathbb{N} \to \mathbb{N}$ such that, for each $n \in \mathbb{N}$ and key k output by $\mathsf{Gen}(1^n)$, F_k is a function with domain either $\{0,1\}^{L(n)}$ or $\{0,1\}^{\leq L(n)}$ and range $\{0,1\}^{\ell(n)}$.
- There exists a deterministic polynomial-time algorithm that outputs $F_k(x)$ when given a key k output by $\mathsf{Gen}(1^n)$ and a string x in the domain of F_k. (For simplicity, we denote this algorithm also by F.)

Definition 43.3. A **family of hash functions** is a family of functions (Gen, H) such that, for each $n \in \mathbb{N}$ and key k output by $\mathsf{Gen}(1^n)$, the function H_k is compressing.

Definition 43.4. A family of hash functions (Gen, H) is said to be **collision resistant** if every polynomial-time attacker $\mathcal{A}$ wins the following security game with at most negligible probability:

1. The challenger runs $\mathsf{Gen}(1^n)$ to generate a key k.
2. The challenger gives k to $\mathcal{A}$.
3. The attacker $\mathcal{A}$ outputs two strings x and x' in the domain of H_k.
4. The attacker $\mathcal{A}$ wins the security game if and only if $x \neq x'$ and $H_k(x) = H_k(x')$.

Note on terminology: For brevity, we often omit the word "family" and refer to a collision-resistant family of hash functions (Gen, H) as a **collision-resistant hash function** or a CRHF for short. We also refer to H and H_k as collision-resistant hash functions, but this practice should be regarded as an informal shorthand only. To define a CRHF, both the algorithms Gen and H must be specified.

Remark: The algorithms Gen and H are assumed to be public and known to the attacker in keeping with Kerckhoffs' principle against security by obscurity. Also, the attacker is given the key in the security game for collision-resistant hash functions; this differs from the case for pseudorandom functions.

It is easy to achieve either collision resistance or compression in isolation: the difficulty is to achieve both simultaneously. For example, to achieve collision resistance without compression, we can simply choose $H_k : \{0,1\}^n \to \{0,1\}^n$ to be the identity function, that is, the function that maps each string to itself. To achieve compression without collision resistance, we instead choose $H_k : \{0,1\}^{n+1} \to \{0,1\}^n$ to be the function that truncates each string in $\{0,1\}^{n+1}$ to its first n bits.

[3] We also denote the size of a set S by $|S|$ and the absolute value of a real number a by $|a|$. These notations should not be confused with the length $|x|$ of a string x.

[4] Some authors would use the term "codomain" here instead of "range".

44 Birthday attack

The birthday attack is a exponential-time attack against an arbitrary hash function. The name of the attack comes from the following problem.

Birthday problem: Suppose we have a class of students. Assume that no student is born during a leap year and that every student is equally likely to be born on each of the 365 days of the year. Also, assume the birthdays of the students are independent of each other. How many students do we need for the probability that at least two students have the same birthday to be $1/2$ or greater? The answer is 23. To see this, let $y_1, \ldots, y_{23}$ denote the birthdays of the first 23 students. There are 365^{23} choices for $(y_1, \ldots, y_{23})$ in total and $365 \cdot 364 \cdots 344 \cdot 343$ choices for $(y_1, \ldots, y_{23})$ in which each birthday is distinct. Therefore, the probability that the birthdays are all distinct is

$$\frac{365 \cdot 364 \cdots 344 \cdot 343}{365^{23}} \leq 0.493,$$

so the probability that at least two birthdays are the same is at least $1/2$.

Now, consider an arbitrary function $f : \{0,1\}^L \to \{0,1\}^\ell$ such that $L > \ell$. The evaluation of f on a specific input is called a **trial**. By the pigeonhole principle, there must exist a collision for f. How may trials does it take to find a collision? We can find a collision in 2^L trials by iterating over all strings in $\{0,1\}^L$ until we find two distinct strings that collide. Can we do better than 2^L trials? Consider the following attack.

Algorithm: Birthday attack

Choose $x_1, \ldots, x_q$ uniformly at random from $\{0,1\}^L$.

for $i := 1$ **to** q **do**
  $y_i := f(x_i)$

for $i := 1$ **to** q **do**
  for $j := 1$ **to** $i - 1$ **do**
    if $y_i = y_j$ **then return** (x_i, x_j)

It can be shown that the birthday attack finds a collision with constant probability when $q = \Theta(2^{\ell/2})$. We shall prove this only for the special case that f is regular.

Definition 44.1. Let $f : D \to R$ be any function. For each $y \in R$, the **pre-image** of y under f is defined to be the set

$$f^{-1}(y) := \{x \in D : f(x) = y\}.$$

We say that f is **regular** if the pre-image of every element in its range has the same size.

The next lemma will help us to bound the number of trials required by the birthday attack.

Lemma 44.2. *Let $y_1, \ldots, y_q$ be chosen independently and uniformly at random from a set of size N. Let P denote the probability that there exist distinct $i, j \in [q]$ such that $y_i = y_j$. If $q^2 \leq 2N$, then*

$$\left(1 - \frac{1}{e}\right)\binom{q}{2}\frac{1}{N} \leq P \leq \binom{q}{2}\frac{1}{N}.$$

Proof. Since the random variables $y_1, \ldots, y_q$ are independent and uniformly distributed on a set of size N, we know that $\Pr[y_i = y_j] = 1/N$ for all distinct $i, j \in [q]$. There are $\binom{q}{2}$ pairs $(i, j) \in [q] \times [q]$ with $i < j$. Thus, by the union bound,

$$P = \Pr\left[\bigvee_{i<j}(y_i = y_j)\right] \leq \sum_{i<j}\Pr[y_i = y_j] = \binom{q}{2}\frac{1}{N}.$$

This proves the upper bound. Now, we prove the lower bound. The probability that $y_1, \ldots, y_q$ are all distinct is $1 - P$. We can compute this probability in another way. There are N^q choices for $(y_1, \ldots, y_q)$ in total and

$N(N-1)\cdots(N-q+1)$ choices for $(y_1,\ldots,y_q)$ in which each term is distinct. Therefore, the probability that $y_1,\ldots,y_q$ are all distinct is

$$\frac{N(N-1)\cdots(N-q+1)}{N^q} = \prod_{k=0}^{q-1}\left(1-\frac{k}{N}\right).$$

Since $1-z \le e^{-z}$ for all $z \in \mathbb{R}$, it follows that

$$1-P = \prod_{k=0}^{q-1}\left(1-\frac{k}{N}\right) \le \exp\left(-\sum_{k=0}^{q-1}\frac{k}{N}\right) = \exp\left(\frac{q(q-1)}{2N}\right).$$

Since $e^{-z} \le 1-(1-1/e)z$ for all $z \in [0,1]$, it follows that if $q^2 \le 2N$, then

$$1-P \le \exp\left(\frac{q(q-1)}{2N}\right) \le 1-\left(1-\frac{1}{e}\right)\binom{q}{2}\frac{1}{N}.$$

The lower bound now follows directly from the above inequality. □

Theorem 44.3. *Let $f : \{0,1\}^L \to \{0,1\}^\ell$ be a regular function such that $L > \ell$. Let $x_1,\ldots,x_q$ be chosen independently and uniformly at random from $\{0,1\}^L$. If $q^2 \le 2^{\ell+1}$, then the probability that at least one of the pairs (x_i,x_j) with $i \ne j$ is a collision for f is at least $(1/2-1/e)\binom{q}{2}2^{-\ell}$.*

Proof. Assume that $q \le 2^{\ell+1}$. For each $i \in [q]$ and $y \in \{0,1\}^\ell$, since x_i is uniformly distributed on $\{0,1\}^L$, we know that

$$\Pr[f(x_i)=y] = \Pr\big[x_i \in f^{-1}(y)\big] = \frac{\big|f^{-1}(y)\big|}{2^L}.$$

But since f is regular, the above probability must be the same for every $y \in \{0,1\}^\ell$. This implies that $f(x_1),\ldots,f(x_q)$ are independently and uniformly distributed on $\{0,1\}^\ell$. Let $\mathcal{E}_1$ denote the event that at least two of the strings $f(x_1),\ldots,f(x_q)$ are the same. Let $\mathcal{E}_2$ denote the event that at least two of the strings $x_1,\ldots,x_q$ are the same. Let $\mathcal{E}$ denote the event that some pair (x_i,x_j) with $i \ne j$ is a collision for f. Note that $\mathcal{E}_1 \wedge \neg\,\mathcal{E}_2 \Rightarrow \mathcal{E}$. Hence, by Lemma 44.2,

$$\begin{aligned}
\Pr[\mathcal{E}] &\ge \Pr[\mathcal{E}_1 \wedge \neg\,\mathcal{E}_2] \\
&\ge \Pr[\mathcal{E}_1] - \Pr[\mathcal{E}_2] \\
&\ge \left(1-\frac{1}{e}\right)\binom{q}{2}2^{-\ell} - \binom{q}{2}2^{-L} \\
&\ge \left(1-\frac{1}{e}\right)\binom{q}{2}2^{-\ell} - \binom{q}{2}2^{-\ell+1} \\
&= \left(\frac{1}{2}-\frac{1}{e}\right)\binom{q}{2}2^{-\ell}.
\end{aligned}$$

□

Theorem 44.3 shows that if $f : \{0,1\}^L \to \{0,1\}^\ell$ is a regular hash function, then the birthday attack finds a collision for f with constant probability in $q = \Theta(2^{\ell/2})$ trials.

45 Merkle–Damgård transform

The Merkle–Damgård transform constructs a CRHF that compresses many bits from a CRHF that compresses only a few bits. Let (Gen, h) be a collision-resistant hash function such that $h_k : \{0,1\}^{2n} \to \{0,1\}^n$ for every $n \in \mathbb{N}$ and key k output by $\mathsf{Gen}(1^n)$. We shall construct a collision-resistant hash function (Gen, H) such that $H_k : \{0,1\}^{<2^n} \to \{0,1\}^n$ for every $n \in \mathbb{N}$ and key k output by $\mathsf{Gen}(1^n)$. The construction is shown on the next page.

Merkle–Damgård transform:

- The key-generation algorithm for H is the same as the key-generation algorithm Gen for h.
- Given a key k output by $\mathsf{Gen}(1^n)$ and a string $x \in \{0,1\}^{<2^n}$, the algorithm H computes $H_k(x)$ as follows:
  1. Let $B := \lceil |x|/n \rceil$.
  2. Pad x with zeros until its length is multiple of n and then divide the resulting string into B blocks $x_1, \ldots, x_B$ of length n.
  3. Let x_{B+1} be the binary representation of $|x|$.
  4. Let $z_0 := 0^n$.
  5. For $i := 1$ to $B+1$, let $y_i := z_{i-1} \,\|\, x_i$ and $z_i := h_k(y_i)$.
  6. Return z_{B+1}.

The hash function h_k used in the Merkle–Damgård transform is often called a **compression function**.

Algorithm H makes $O(|x|/n)$ calls to h. Since h runs in time polynomial in n and $|x|$ and all other computation done by H can be performed in time polynomial in n and $|x|$, it follows that H runs in time polynomial in n and $|x|$.

Theorem 45.1. *If* (Gen, h) *is collision resistant, then* (Gen, H) *is also collision resistant.*

Proof. Suppose (Gen, h) is collision resistant, and let $\mathcal{A}$ be a polynomial-time attacker that wins the security game against (Gen, H) with non-negligible probability. We construct an algorithm $\mathcal{B}$ attacking (Gen, h) as follows.

algorithm $\mathcal{B}$:

1. Receive the key k from the challenger and give k to $\mathcal{A}$.
2. Receive a pair of strings (x, x') from $\mathcal{A}$.
3. If $x = x'$ or $H_k(x) \neq H_k(x')$, then abort.
4. Compute B, $x_1, \ldots, x_{B+1}$, $y_1, \ldots, y_{B+1}$, and $z_0, \ldots, z_{B+1}$ as described above in the Merkle–Damgård transform. Similarly compute B', $x'_1, \ldots, x'_{B'+1}$, $y'_1, \ldots, y'_{B'+1}$, and $z'_0, \ldots, z'_{B'+1}$ for x'.
5. If $y_{B+1} \neq y'_{B'+1}$, then return $(y_{B+1}, y'_{B'+1})$.
6. Otherwise, find the largest $j \in [B]$ such that $y_j \neq y'_j$. Return (y_j, y'_j).

Algorithm $\mathcal{B}$ runs $\mathcal{A}$ and makes polynomially many calls to h. Since $\mathcal{A}$ and h both run in polynomial time and all other computation done by $\mathcal{B}$ can be performed in polynomial time, it follows that $\mathcal{B}$ runs in polynomial time. Next, we prove that whenever $\mathcal{A}$ outputs a collision for H, algorithm $\mathcal{B}$ outputs a collision for h. This implies that the probability that $\mathcal{B}$ outputs a collision for h is at least as large as the non-negligible probability that $\mathcal{A}$ outputs a collision for H.

Suppose $\mathcal{A}$ outputs a collision (x, x') for H. In that case, algorithm $\mathcal{B}$ does not abort. Since (x, x') is a collision for H, we know that

$$h_k(y_{B+1}) = z_{B+1} = H_k(x) = H_k(x') = z'_{B'+1} = h_k(y'_{B'+1}).$$

If $y_{B+1} \neq y_{B'+1}$, then $\mathcal{B}$ outputs $(y_{B+1}, y'_{B'+1})$, which must be a collision for h by the above equality. Now, assume that $y_{B+1} = y'_{B'+1}$. This implies $z_B = z'_{B'}$ and $x_{B+1} = x'_{B'+1}$. Since x_{B+1} and $x'_{B'+1}$ are respectively the lengths of x and x' in binary, it follows that $|x| = |x'|$ and hence $B = \lceil |x|/n \rceil = \lceil |x'|/n \rceil = B'$. Since (x, x') is a collision, we know that $x \neq x'$. Hence, there must exist an index $i \in [B]$ such that $x_i \neq x'_i$, which implies there exists an index $i \in [B]$ such that $y_i \neq y'_i$. By definition, j is the largest such index. Therefore, $y_{j+1} = y'_{j+1}$ and hence $z_j = z'_j$. Since $y_j \neq y'_j$ and $h_k(y_j) = z_j = z'_j = h_k(y'_j)$, it follows that (y_j, y'_j) is a collision for h. Thus, $\mathcal{B}$ outputs a collision for h whenever $\mathcal{A}$ outputs a collision for H. □

46 Hash functions in practice

In this section, we discuss some hash functions that are commonly used in practice. Strictly speaking, none of these functions fit the formal definition of collision-resistant hash functions, because they are not parameterized by a key.

The hash function MD5 was designed by Ronald Rivest in 1991 to address weaknesses in MD4, a hash function designed by Rivest in 1990. SHA-0 was published by NIST in 1993 but was withdrawn shortly after publication, because the NSA found a flaw, which they never disclosed. SHA-1 was designed by the NSA to replace SHA-0 and was published by NIST in 1995. The design of MD5 was based on that of MD4, and the design of SHA-1 was based those of MD4 and MD5. The NSA also designed SHA-2, a set of four hash functions: SHA-224, SHA-256, SHA-384, and SHA-512. NIST published SHA-256, SHA-384, and SHA-512 in 2001 and SHA-224 in 2004. NIST is currently holding a competition[5] to determine the hash function that will become the new standard for SHA-3. Currently, there are five finalists, and the winner is scheduled to be selected by the end of 2012.

MD5, SHA-1, and the SHA-2 hash functions are constructed using a version of the Merkle–Damgård transform that uses a compression function with a fixed key. (Recall that the formal definition of CRHFs requires that the key be a parameter.) The table below shows input and output lengths, as well as the block size and compression function output length, for the hash functions discussed above. The lengths below are given in bits.

	maximum input length	block size	compression function output length	final output length
MD5	$2^{64} - 1$	512	128	128
SHA-1	$2^{64} - 1$	512	160	160
SHA-224	$2^{64} - 1$	512	256	224
SHA-256	$2^{64} - 1$	512	256	256
SHA-384	$2^{128} - 1$	1024	512	384
SHA-512	$2^{128} - 1$	1024	512	512

MD5 is now known to be insecure and should not be used in applications that require collision resistance. In 2004, Xiaoyun Wang, Dengguo Feng, Xuejia Lai, and Hongbo Yu presented an attack that finds collisions for MD5 in approximately 2^{39} trials; in comparison, the birthday attack finds a collision in approximately $2^{128/2} = 2^{64}$ trials. In 2005, Arjen Lenstra, Xiaoyun Wang, and Benne de Weger used the attack described by Wang et al. to produce two different X.509 certificates with the same signature.

47 Other types of hash functions

In this section, we define some other types of hash functions, but we shall not discuss them in detail.

Definition 47.1. A family of functions (Gen, F) is said to be **one-way** if every polynomial-time attacker $\mathcal{A}$ wins the following security game with at most negligible probability:

1. The challenger runs $\mathsf{Gen}(1^n)$ to generate a key k.
2. The challenger chooses x uniformly at random from the domain of F_k.
3. The challenger computes $y := F_k(x)$.
4. The challenger gives k and y to $\mathcal{A}$.
5. The attacker $\mathcal{A}$ outputs a string x' in the domain of F_k.
6. The attacker $\mathcal{A}$ wins the security game if and only if $F_k(x') = y$.

Remark: To win the security game above, the attacker may output *any* string x' in the domain of F_k such that $F_k(x') = y$; the string x' need not be the same as x.

Definition 47.2. A family of hash functions (Gen, H) is said to be **pre-image resistant** if it is one-way.

[5] `http://csrc.nist.gov/groups/ST/hash/sha-3/index.html`

Definition 47.3. A family of hash functions (Gen, H) is said to be **second-pre-image resistant** if every polynomial-time algorithm $\mathcal{A}$ wins the following security game with at most negligible probability:

1. The challenger runs $\mathsf{Gen}(1^n)$ to generate a key k.
2. The challenger chooses x uniformly at random from the domain of H_k.
3. The challenger gives k and x to $\mathcal{A}$.
4. The attacker $\mathcal{A}$ outputs a string x' in the domain of H_k.
5. The attacker $\mathcal{A}$ wins the security game if and only if $x \neq x'$ and $H_k(x) = H_k(x')$.

Definition 47.4. A **universal one-way family of hash functions** is a family of hash functions (Gen, H) such that every polynomial-time attacker $\mathcal{A}$ wins the security game below with at most negligible probability:

1. The challenger runs $\mathsf{Gen}(1^n)$ to generate a key k.
2. The challenger gives 1^n to $\mathcal{A}$.
3. The attacker $\mathcal{A}$ outputs a string x in the domain of H_k (before seeing the key k).
4. The challenger gives k to $\mathcal{A}$.
5. The attacker $\mathcal{A}$ outputs a string x' in the domain of H_k (after seeing the key k).
6. The attacker $\mathcal{A}$ wins the security game if and only if $x \neq x'$ and $H_k(x) = H_k(x')$.

Remark: The reason the challenger gives 1^n to $\mathcal{A}$ in step 2 of the above security game is so that $\mathcal{A}$ can determine the size of the domain of H_k before seeing the key k.

Note on terminology: For brevity, we often omit the word "family" and refer to a pre-image resistant family of hash functions as a **pre-image resistant hash function**; we also refer to a second-pre-image resistant family of hash functions as a **second-pre-image resistant hash function**, and we refer to a universal one-way family of hash functions as a **universal one-way hash function** or a UOWHF for short.

We may also refer to H or H_k as a pre-image resistant hash function, second-pre-image resistant hash function, or universal one-way hash function, but this practice should be regarded as an informal shorthand only. Both the algorithms Gen and H must be specified in a formal definition.

Remark: Every CRHF is a UOWHF, and every UOWHF is a second-pre-image resistant hash function. Your textbook claims that every second-pre-image resistant hash function is also pre-image resistant, but this claim is only conditionally true: it depends on how much larger the domain is than the range.

48 Introduction

The goal of this lecture is to construct a collision-resistant hash function based on the discrete logarithm assumption. Recall, from previous lectures, that we defined the group-generation algorithm $\mathcal{G}$ as follows.

algorithm $\mathcal{G}$:

1. Take 1^n as input (where n is the security parameter).
2. Randomly choose an $(n+1)$-bit prime p of the form $p = 2q + 1$, where q is also prime.
3. Let $\mathbb{G}$ denote the subgroup of $\mathbb{Z}_p^*$ of order q.
4. Randomly choose a generator g of $\mathbb{G}$.
5. Output (p, g).

We shall call $\mathcal{G}$ in the key-generation algorithm of our collision-resistant hash function.

49 Collision resistance from discrete-log

Our collision-resistant hash function is defined as follows.

function $\mathsf{Gen}(1^n)$
Run $\mathcal{G}(1^n)$ to generate (p, g). $q := (p-1)/2$ Choose $a \in \mathbb{Z}_q$ uniformly at random. $h := g^a$
$k := (p, g, h)$ **return** k
// The domain of H_k is $\mathbb{Z}_q \times \mathbb{Z}_q$, and the range of H_k is $\mathbb{G}$.
function $H_k(x)$
$(p, g, h) := k$ $(x_1, x_2) := x$ **return** $g^{x_1} h^{x_2}$

The algorithms Gen and H both run in polynomial time, because $\mathcal{G}$ runs in polynomial time and because operations in $\mathbb{Z}_p^*$ can be performed in polynomial time. Strictly speaking, (Gen, H) does not satisfy the definition of a CRHF given in previous lectures, because the inputs and outputs of H_k are not bit-strings. This technicality can be overcome by choosing an efficient encoding of the elements of $\mathbb{Z}_q \times \mathbb{Z}_q$ and $\mathbb{G}$ as bit-strings. In such an encoding, the inputs to H_k should be roughly twice as long as the outputs of H_k, because $\mathbb{Z}_q$ has the same size as $\mathbb{G}$. In other words, with a proper encoding, the function H_k is compressing.

Theorem 49.1. *If the discrete logarithm assumption is true for $\mathcal{G}$, then (Gen, H) is collision resistant.*

Proof. We shall prove the contrapositive: if (Gen, H) is not collision resistant, then the DL assumption for $\mathcal{G}$ is false. Recall that the DL challenger runs $\mathcal{G}(1^n)$ to generate (p, g), chooses $a \in \mathbb{Z}_q$ uniformly at random, where $q = (p-1)/2$, and finally gives (p, g, g^a) to the attacker.

Let $\mathcal{A}$ be a polynomial-time algorithm attacking (Gen, H) that finds a collision for H with non-negligible probability. We construct an algorithm $\mathcal{B}$ for solving the DL problem as follows.

algorithm $\mathcal{B}$:

1. Receive (p, g, g^a) from the DL challenger. (Note that $\mathcal{B}$ does not know a.)
2. Let $k := (p, g, g^a)$ and $q := (p-1)/2$.
3. Give k to $\mathcal{A}$ and receive (x, x') in response.
4. If $x = x'$ or $H_k(x) \neq H_k(x')$, then abort.
5. Parse x and x' respectively as ordered pairs (x_1, x_2) and (x_1', x_2') in $\mathbb{Z}_q \times \mathbb{Z}_q$.
6. Output $(x_1 - x_1')(x_2' - x_2)^{-1}$.

Algorithm $\mathcal{B}$ runs in polynomial time, because $\mathcal{A}$ and H run in polynomial time and because operations in $\mathbb{Z}_q$ can be performed in polynomial time. Next, we prove that whenever $\mathcal{A}$ outputs a collision for H, algorithm $\mathcal{B}$ outputs the correct answer to the DL problem. This implies that the probability that $\mathcal{B}$ correctly solves the DL problem is at least as large as the non-negligible probability that $\mathcal{A}$ finds a collision for H.

Suppose $\mathcal{A}$ outputs a collision (x, x') for H. In that case, algorithm $\mathcal{B}$ does not abort. Since (x, x') is a collision for H, we know that

$$g^{x_1} h^{x_2} = H_k(x) = H_k(x') = g^{x_1'} h^{x_2'}$$

and hence

$$g^{x_1 - x_1'} = h^{x_2' - x_2}. \tag{49.1}$$

Suppose, for the moment, that $x_2 = x_2'$. Then $g^{x_1 - x_1'} = h^{x_2' - x_2} = 1$, which implies $x_1 = x_1'$. But if $x_1 = x_1'$ and $x_2 = x_2'$, then $x = x'$, which is impossible because (x, x') is a collision. Thus, x_2 and x_2' must be distinct.

Since $x_2 \neq x_2'$, we know that $x_2' - x_2 \neq 0$, so algorithm $\mathcal{B}$ can compute $(x_2' - x_2)^{-1}$. We conclude from (49.1) that

$$g^{(x_1 - x_1')(x_2' - x_2)^{-1}} = h^{(x_2' - x_2)(x_2' - x_2)^{-1}} = h = g^a,$$

which implies $(x_1 - x_1')(x_2' - x_2)^{-1} = a$. Thus, $\mathcal{B}$ outputs the correct answer to the DL problem. □

50 Public-key encryption

We shall describe the El Gamal public-key encryption scheme and prove that it is IND-CPA secure. But we must first define public-key encryption schemes and define what it means for a public-key encryption scheme to be IND-CPA secure.

Definition 50.1. A **public-key encryption scheme** with **message space** $\mathcal{M}$ and **ciphertext space** $\mathcal{C}$ is an ordered triple $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ of probabilistic polynomial-time algorithms satisfying the conditions below:

- Gen takes 1^n as input (where n is the security parameter) and outputs a key (pk, sk). We call pk the **public key** and call sk the **secret key** or **private key**.
- Enc takes a public key pk and a message $m \in \mathcal{M}$ as input and outputs a ciphertext $c \in \mathcal{C}$.
- Dec takes a secret key sk and a ciphertext $c \in \mathcal{C}$ as input and outputs a message $m \in \mathcal{M}$.

The **key space** $\mathcal{K}$ is the set of all keys output by $\mathsf{Gen}(1^n)$. We say that $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is **correct** if

$$\mathsf{Dec}\big(sk, \mathsf{Enc}(pk, m)\big) = m \quad \text{for all } (pk, sk) \in \mathcal{K} \text{ and } m \in \mathcal{M}.$$

Definition 50.2. Let $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a public-key encryption scheme. We say that $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ has **indistinguishable encryptions under chosen-plaintext attack** (or is **IND-CPA secure**, for short) if every polynomial-time attacker $\mathcal{A}$ wins the following security game with at most negligible advantage:

1. The challenger runs $\mathsf{Gen}(1^n)$ to generate (pk, sk).
2. The challenger gives 1^n and the public key pk to $\mathcal{A}$.
3. The attacker $\mathcal{A}$ chooses two messages m_0 and m_1 of equal length and gives them to the challenger.
4. The challenger chooses $b \in \{0, 1\}$ uniformly at random.
5. The challenger computes $c^* := \mathsf{Enc}(pk, m_b)$ and gives c^* to $\mathcal{A}$.
6. The attacker $\mathcal{A}$ outputs $b' \in \{0, 1\}$.
7. The attacker $\mathcal{A}$ wins the security game if and only if $b = b'$.
8. The advantage of $\mathcal{A}$ is defined as $\Pr[b = b'] - 1/2$.

Remark: The algorithms Gen, Enc, and Dec are assumed to be public and known to the attacker in keeping with Kerckhoffs' principle against security by obscurity.

Remark: Recall that in the IND-CPA security game for *private-key encryption*, there are two query phases in which the attacker is given oracle access to the encryption function. No query phases are needed in the IND-CPA security game for *public-key encryption*, because the attacker is given the public key and thus can encrypt messages itself.

Remark: A public-key encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ in which Enc is deterministic cannot be IND-CPA secure, because the attacker can choose two distinct messages m_0 and m_1 of equal length, compute ciphertexts $c_0 := \mathsf{Enc}(pk, m_0)$ and $c_1 := \mathsf{Enc}(pk, m_1)$, and then compare c_0 and c_1 with the challenge ciphertext c^*.

51 The El Gamal encryption scheme

In this section, we describe the El Gamal encryption scheme and prove that it is IND-CPA secure under the DDH assumption. Recall that in previous lectures, we defined the group-generation algorithm $\mathcal{G}$ as follows.

algorithm $\mathcal{G}$:

1. Take 1^n as input (where n is the security parameter).
2. Randomly choose an $(n+1)$-bit prime p of the form $p = 2q + 1$, where q is also prime.
3. Let $\mathbb{G}$ denote the subgroup of $\mathbb{Z}_p^*$ of order q.
4. Randomly choose a generator g of $\mathbb{G}$.
5. Output (p, g).

The El Gamal encryption scheme is defined as follows.

---

function $\mathsf{Gen}(1^n)$
Run $\mathcal{G}(1^n)$ to generate (p, g). $q := (p-1)/2$ Choose $x \in \mathbb{Z}_q$ uniformly at random. $h := g^x$ $pk := (p, g, h)$ $sk := (p, g, x)$ **return** (pk, sk)

function $\mathsf{Enc}(pk, m)$
$(p, g, h) := pk$ $q := (p-1)/2$ Choose $y \in \mathbb{Z}_q$ uniformly at random. $c_1 := g^y$, $c_2 := h^y m$ **return** (c_1, c_2)

function $\mathsf{Dec}(sk, c)$
$(p, g, x) := sk$ $(c_1, c_2) := c$ **return** $c_1^{-x} c_2$

The algorithms Gen, Enc, and Dec run in polynomial time, because $\mathcal{G}$ runs in polynomial time and because operations in $\mathbb{Z}_p^*$ can be performed in polynomial time. Let $m \in \mathbb{G}$ and suppose (pk, sk) is a key output by $\mathsf{Gen}(1^n)$. Let x, h, and y be as described above. Then

$$\mathsf{Dec}\big(sk, \mathsf{Enc}(pk, m)\big) = \mathsf{Dec}\big(sk, (g^y, h^y m)\big) = (g^y)^{-x} h^y m = (g^y)^{-x} (g^x)^y m = m.$$

Thus, $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a correct public-key encryption scheme.

Theorem 51.1. *If the decisional Diffie-Hellman assumption is true for $\mathcal{G}$, then the El Gamal encryption scheme is IND-CPA secure.*

Proof. Assume that the DDH assumption is true for $\mathcal{G}$. Recall that the DDH challenger works as follows: runs $\mathcal{G}(1^n)$ to generate (p, g); chooses $x, y, z \in \mathbb{Z}_q$ uniformly at random, where $q = (p-1)/2$; chooses $d \in \{0, 1\}$ uniformly at random; sets $T := g^{xy}$ if $d = 0$ and $T := g^z$ if $d = 1$; and finally, gives (p, g^x, g^y, T) to the attacker.

Let $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be the El Gamal encryption scheme. Let $\mathcal{A}$ be a polynomial-time algorithm attacking Π. Let ε denote the advantage of $\mathcal{A}$ in the IND-CPA security game against Π. We construct an algorithm $\mathcal{B}$ for solving the DDH problem as follows.

---

algorithm $\mathcal{B}$:

1. Receive (p, g, g^x, g^y, T) from the DDH challenger. (Note that $\mathcal{B}$ knows neither x nor y.)
2. Let $q := (p-1)/2$ and let n be the length of q in bits.
3. Let $pk := (p, g, g^x)$.
4. Give 1^n and pk to $\mathcal{A}$.
5. Receive messages m_0 and m_1 from $\mathcal{A}$.
6. Choose $b \in \{0, 1\}$ uniformly at random.
7. Let $c^* := (g^y, T m_b)$ and give c^* to $\mathcal{A}$.
8. Let b' denote the guess output by $\mathcal{A}$.
9. If $b = b'$, then set $d' := 0$. If $b \neq b'$, then set $d' := 1$.
10. Output d'.

Algorithm $\mathcal{B}$ runs in polynomial time, because $\mathcal{A}$ runs in polynomial time and because operations in $\mathbb{Z}_p^*$ can be performed in polynomial time. The probability that $\mathcal{B}$ wins the DDH security game is

$$\begin{aligned}\Pr[d = d'] &= \Pr[d = 0]\ \Pr[d = d' \mid d = 0] + \Pr[d = 1]\ \Pr[d = d' \mid d = 1] \\ &= \frac{1}{2}\Pr[d' = 0 \mid d = 0] + \frac{1}{2}\Pr[d' = 1 \mid d = 1] \\ &= \frac{1}{2}\Pr[b = b' \mid d = 0] + \frac{1}{2}\Pr[b \neq b' \mid d = 1]. \end{aligned} \tag{51.1}$$

When $d = 0$, the DDH challenger sets $T := g^{xy}$, so the view that $\mathcal{B}$ presents to $\mathcal{A}$ is identical to the actual IND-CPA security game against Π. Therefore, the probability that $b = b'$ given $d = 0$ is the same as the probability that $\mathcal{A}$ wins the IND-CPA security game against Π; in other words,

$$\Pr[b = b' \mid d = 0] = \frac{1}{2} + \varepsilon. \tag{51.2}$$

When $d = 1$, the DDH challenger sets $T := g^z$. Recall that $\mathbb{G}$ denotes the subgroup of $\mathbb{Z}_p^*$ of order q. Since z is uniformly distributed in $\mathbb{Z}_q$, it follows that $g^z m_b$ is uniformly distributed in the group $\mathbb{G}$, independently of g, m_0, m_1, and b. Moreover, the random variables g, g^x, g^y, $g^z m_b$, and b are jointly independent. Hence, pk and c^* reveal no information about b, so the guess b' output by $\mathcal{A}$ must be independent of b. Since b is either 0 or 1, each with probability $1/2$, it follows that

$$\Pr[b \neq b' \mid d = 1] = \frac{1}{2}. \tag{51.3}$$

It follows from (51.1), (51.2), and (51.3) that

$$\Pr[d = d'] = \frac{1}{2}\left(\frac{1}{2} + \varepsilon\right) + \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{2} + \frac{\varepsilon}{2}.$$

Thus, $\mathcal{B}$ wins the DDH security game with advantage $\varepsilon/2$. By DDH assumption, algorithm $\mathcal{B}$ can win the DDH security game with only negligible advantage, so $\varepsilon/2$ must be negligible. This implies that ε is also negligible. Therefore, algorithm $\mathcal{A}$ has only negligible advantage ε in the IND-CPA game against Π. □

52 Introduction

In this lecture we will be concerned with the problem of message integrity in the public key setting. We shall introduce the concept of digital signature schemes and their correctness and security requirements. In the last section we will see some concepts from group theory, which will be used in several digital signature constructions.

53 Secure Communication

Secure communication over a public channel encompasses two *different* security goals:

- **Privacy:** This goal, which is also referred to as confidentiality, means that no information about the message is allowed to be obtained from unwanted parties that listen to the public channel. This is achieved by encryption functionalities, which make the retrieval of information from the transmitted message infeasible.
- **Authenticity:** This goal means that the receiver of the message is able to verify that it has not been altered in any way during the transfer over the public channel. This is achieved by MAC schemes (symmetric key setting) or digital signature schemes (asymmetric key setting), which ensure that the tampering of the transmitted message without being detected is infeasible.

Cryptographic schemes for achieving the above goals have been proposed in both the symmetric and the asymmetric key settings. The four related cryptographic primitives are shown in figure 11. In this lecture we will focus on the bottom right part: the digital signature schemes.

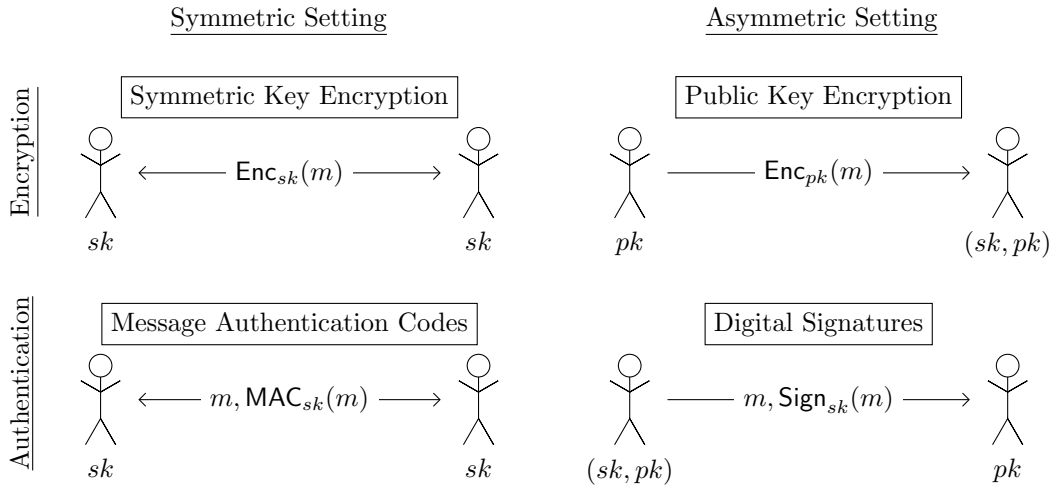

Figure 11: Secure Communication Functionalities

Applications

We mention here that *authenticity* is important in various situations such as:

- Downloading of software patches
- Financial transactions, stock market
- Military commands
- Email communications
- Etc.

Relation of digital signature schemes to MAC schemes

As one can see in figure 11, in digital signature schemes *only* the signer, who holds the public key pk and the secret key sk [6], is able to produce a signature $\sigma = \mathsf{Sign}_{sk}(m)$. The verifier on the other hand can verify that the signature is valid using only the public key. Therefore, digital signatures are publicly verifiable by anyone. Thus compared to MAC schemes, they are:

- **Transferable:** Anyone can verify their validity without compromising the future usability of the scheme, where in MAC schemes in order to verify one needs the secret key. For example, they can be used in a contract signed by several parties to be shown to a judge.
- **Non - Repudiable:** The signer can not claim that he did not sign the message, since he is the only one in possession of the secret key.

Relation of digital signature schemes to encryption schemes

Encryption is a *different* security goal than authentication. As a result it is possible that an encryption scheme does not guarantee any form of authentication on its own. It might be the case that although the system is secure in the sense that no information is leaked about the message, an active adversary can transform the ciphertext to something encrypting a different (possibly random) message; thus breaking authentication.

Also, extra care has to be taken on the way of combining an encryption system to a signing one, because their combination might be insecure (although both schemes are secure in their respective definitions). See section 4.9, page 148, on this topic.

Finally, signature schemes are *not* the "opposite" of encryption schemes. There is not always a meaningful way to construct a signature scheme from an encryption scheme by "reversing" it and even when this is possible, the resulting scheme is insecure.

Short History of Digital Signatures

1976 : Introduction of the concept by Diffie and Hellman.
1978 : Rivest, Shamir, and Adleman design RSA signatures.
1984 : Strict definitions of security by Goldwasser, Micali, and Rivest (GMR) and by Goldreich. Initially thought impossible to achieve.
1993 : Digital Signature Algorithm (DSA) adopted by NIST with patent filed by Kravitz (NSA). The scheme is royalty-free, but no security proof is given.
2000 : Bill Clinton signs E-SIGN Act.
2002 : RSA wins Turing Award.
Present : Billions of certificates verified every day on the Internet (VeriSign / Symantec).

54 Definitions of Digital Signature schemes

Intuition

The main intuition behind the digital signature schemes is the same as the "real life" signatures. That is, they provide a procedure to tag a message with your own signature; one that nobody else can reproduce correctly, even if he has seen many of your signatures. Another requirement is that anyone can verify that your signature is indeed yours by a public procedure.

For the above to be true in the digital communication setting, the signer holds a secret key that allows him (and only him) to sign any message. The public key is used to verify that the signature was created by the signer. The correctness requirement states that whatever the signer signs verifies correctly, while the security requirement asserts that no one else can sign correctly.

Algorithm Specification (Ch. 12.1)

Definition 54.1. A *signature scheme* is a tuple of three probabilistic polynomial-time algorithms $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ that work as follows:

[6] Sometimes the public key pk is called the verification key and the secret key sk is called the signing key.

- $\mathsf{Gen}(1^n) \to (pk, sk)$: The setup algorithm takes in the security parameter $n \in \mathbb{N}$ encoded in unary and outputs the public key pk and the secret key sk of the system.
- $\mathsf{Sign}(sk, m) \to \sigma$: The signing algorithm takes in the secret key sk and a message m belonging to a suitable message space $\mathcal{M}$. It outputs the signature σ. Sometimes we write for convenience $\mathsf{Sign}_{sk}(m)$ instead of $\mathsf{Sign}(sk, m)$.
- $\mathsf{Vrfy}(pk, m, \sigma) \to \{0, 1\}$: The verification algorithm takes in the public key pk, the message m and the signature σ. It outputs either $\mathsf{invalid}/\mathsf{reject} \to 0$ or $\mathsf{valid}/\mathsf{accept} \to 1$. As before, we might write $\mathsf{Vrfy}_{pk}(m, \sigma)$ instead of $\mathsf{Vrfy}(pk, m, \sigma)$.

Correctness

It is required that for every $n \in \mathbb{N}$, every pair of keys (pk, sk) output by $\mathsf{Gen}(1^n)$, and every message $m \in \mathcal{M}$, it holds that [7]

$$\mathsf{Vrfy}(pk, m, \mathsf{Sign}(sk, m)) = 1$$

Security

Let $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a signature scheme and consider the following experiment for an adversary $\mathcal{A}$ and security parameter n:

Experiment $\mathsf{Sig\text{-}Forge}_{\mathcal{A},\Pi}(n)$:
1. Run $\mathsf{Gen}(1^n) \to (pk, sk)$
2. Adversary $\mathcal{A}$ is given pk and oracle access to $\mathsf{Sign}_{sk}(\cdot)$. $\mathcal{A}$ then outputs (m, σ).
   Let Q denote the set of messages that $\mathcal{A}$ queried to the oracle.
3. The output of the experiment is defined to be 1 if and only if
   (1) $\mathsf{Vrfy}_{pk}(m, \sigma) = 1$ and
   (2) $m \notin Q$

Definition 54.2 (12.2)**.** A signature scheme $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ is *existentially unforgeable under an adaptive chosen-message attack* if for all probabilistic polynomial-time adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(n)$ such that

$$\Pr[\mathsf{Sig\text{-}Forge}_{\mathcal{A},\Pi}(n) = 1] \leq \mathsf{negl}(n)$$

55 The Algebraic Setting

In this section we review some concepts of group theory that will be used in the construction of signature schemes.

Definition 55.1. A *group* is a set $\mathbb{G}$ along with a binary operator $\circ$ such that the following properties are satisfied:

1. Closure: For any elements $g, h \in \mathbb{G}$ the element $g \circ h$ is in $\mathbb{G}$.
2. Existence of identity: There is an element $e \in \mathbb{G}$, called the identity element, such that for any $g \in \mathbb{G}$ it is true that $e \circ g = g = g \circ e$.
3. Existence of inverses: For any element $g \in \mathbb{G}$ there exists an element $h \in \mathbb{G}$ such that $g \circ h = e = h \circ g$, where e is the identity element.
4. Associativity: For any elements $g_1, g_2, g_3 \in \mathbb{G}$ it is true that $(g_1 \circ g_2) \circ g_3 = g_1 \circ (g_2 \circ g_3)$.

A group is called Abelian if the following property is satisfied as well:

5. Commutativity: For any elements $g, h \in \mathbb{G}$ it is true that $g \circ h = h \circ g$.

We will denote by $|\mathbb{G}|$ the number of elements of the set $\mathbb{G}$, which is called the *order* of the group.

[7] The correctness definition can be relaxed to state that the probability that the verification algorithm outputs 0 is negligible in n; thus allowing for a small error.

The group $\mathbb{Z}_N^*$

To build signature schemes we will use a specific type of groups which consist of integers modulo a natural number $N \in \mathbb{N}$. The set

$$\mathbb{Z}_N = \{0, 1, \ldots, N-1\}$$

of all the integers modulo N is a group with respect to addition modulo N (*additive* group).

However for our purposes we want a group with respect to multiplication modulo N. In that case $\mathbb{Z}_N$ (or even $\mathbb{Z}_N \backslash \{0\}$) is not always a group, since there might be elements with no inverses modulo N. As we know, these are exactly the numbers a such that $\gcd(a, N) \neq 1$. Therefore, if we remove these elements we get the following *multiplicative* group:

$$\mathbb{Z}_N^* \stackrel{\text{def.}}{=} \{a \in \{1, 2, \ldots, N-1\} | \gcd(a, N) = 1\}$$

We define the Euler phi function to be the order of the above group:

$$\varphi(N) \stackrel{\text{def.}}{=} |\mathbb{Z}_N^*|$$

It is proved that if $N = p \cdot q$, where p, q are two prime numbers, then $\varphi(N) = (p-1)(q-1)$ (see page 255).

We show the case of $N = 15$ in the following example, though real signature schemes use (at least) 1024 or 2048-bit numbers for the order of the groups.

Example 55.2. Let $N = 15 = 5 \cdot 3$. Then according to the above $|\mathbb{Z}_{15}^*| = \varphi(15) = 4 \cdot 2 = 8$. Indeed:

$$\mathbb{Z}_{15}^* = \{1, 2, 4, 7, 8, 11, 13, 14\}$$

All elements have inverses with respect to multiplication modulo 15. For example, the inverse of 8 is 2, since $8 \cdot 2 = 16 = 1 \pmod{15}$, while the inverse of 4 is 4, since $4 \cdot 4 = 16 = 1 \pmod{15}$.

In the next lecture we will see "hard" problems in these groups and how to build signature schemes with them.

56 Introduction

In this lecture we will see two assumptions conjectured to hold on the multiplicative $\mathbb{Z}_N^*$ groups and some digital signature constructions.

57 Factoring Assumption

In general, the factoring problem is the problem of finding the prime factors of a composite number N in time polynomial in the size of N's representation, i.e. the number of N's bits. Even for the "easy" case where N is the product of only two prime numbers of relatively the same size, no efficient algorithm has been found, although the problem has been studied for more than 100 years. Today's computers can brute force around 768-bit numbers, at best.

To define the problem we first define a polynomial time algorithm that generates an instance of it, i.e. the composite number N:

$$\mathsf{GenMod}(1^n) \to (N, p, q) \text{ such that } p, q \text{ are } n\text{-bit primes and } N = p \cdot q$$

There are polynomial time (in n) GenMod algorithms, but their description is out of the scope of this lecture. The factoring assumption is defined via the following experiment for an adversary $\mathcal{A}$ and a security parameter $n \in \mathbb{N}$:

Experiment $\mathsf{Factor}_{\mathcal{A},\mathsf{GenMod}}(n)$:
1. Run $\mathsf{GenMod}(1^n)$ to obtain N, such that $N = p \cdot q$.
2. $\mathcal{A}$ is given N and outputs $p', q' > 1$.
3. The output of the experiment is 1 if and only if $p'q' = N$.

Remark: Since N is the product of two primes and $p', q' > 1$, if the experiment outputs 1 we have that either ($p = p'$ and $q = q'$) or ($p = q'$ and $q = p'$). Therefore the adversary has successfully factored N.

Definition 57.1. Factoring is hard relative to GenMod if for all probabilistic polynomial-time algorithms $\mathcal{A}$ there is a negligible function $\mathsf{negl}(n)$ such that

$$\Pr[\mathsf{Factor}_{\mathcal{A},\mathsf{GenMod}}(n) = 1] \leq \mathsf{negl}(n)$$

58 RSA Assumption

Although the factoring assumption is useful for various cryptographic notions, we will use a different but related assumption for the constructions of digital signature schemes. This is the RSA assumption introduced in 1978 by Rivest, Shamir, Adleman. This assumption uses a generator algorithm to generate an instance of the problem, similar to factoring's GenMod:

$$\begin{aligned}&\mathsf{GenRSA}(1^n) \to (N, e, d) \text{ such that}\\ &\quad N = pq, \text{ where } p, q \text{ are } n\text{-bit primes,}\\ &\quad e > 2 \text{ with } \gcd(e, \varphi(N)) = 1 \text{ and}\\ &\quad d \text{ such that } ed = 1 \pmod{\varphi(N)}\end{aligned}$$

The assumption is is defined via the following experiment for an adversary $\mathcal{A}$ and a security parameter $n \in \mathbb{N}$:

Experiment $\mathsf{RSA}_{\mathcal{A},\mathsf{GenRSA}}(n)$:
1. Run $\mathsf{GenRSA}(1^n) \to (N, e, d)$.
2. Choose uniformly at random an element y from the multiplicative group $\mathbb{Z}_N^*$.
3. $\mathcal{A}$ is given (N, e, y) and outputs $x \in \mathbb{Z}_N^*$.
4. The output of the experiment is 1 if and only if $x^e = y \pmod{N}$.

Definition 58.1. The RSA problem is hard relative to GenRSA if for all probabilistic polynomial-time algorithms $\mathcal{A}$ there is a negligible function $\mathsf{negl}(n)$ such that

$$\Pr[\mathsf{RSA}_{\mathcal{A},\mathsf{GenRSA}}(n) = 1] \leq \mathsf{negl}(n)$$

Relation to Factoring

We say that the RSA problem is "easier" than factoring or, equivalently, that the factoring assumption is "weaker" than the RSA assumption, because if the factoring problem is easy to solve, then the RSA problem is also easy to solve. The attack is the following:

1. Factor N using the assumed poly-time algorithm and retrieve p, q.
2. Compute $\varphi(N) = (p-1)(q-1)$.
3. Compute the inverse d of e modulo $\varphi(N)$ using a poly-time algorithm. For example, the extended Euclidean algorithm.
4. Output $y^d \pmod N$.

The above attack solves the RSA problem because $(y^d)^e = y^{de} = y^{de \bmod \varphi(N)} = y \pmod N$, where the second to last equation is the generalized Fermat's little theorem.

Therefore, the RSA problem can be solved through factoring, but there may be some other way to solve it. As with all assumptions, we are not sure that it is indeed true, but let's assume it is hard and try to build digital signatures using it!

59 "Textbook" RSA Scheme

The first signature scheme we will examine is the so-called "textbook" RSA, which consists of the following algorithms:

$\mathsf{Gen}(1^n)$:
1. Run $\mathsf{GenRSA}(1^n) \to (N, e, d)$
2. Output $pk = (N, e)$ and $sk = (N, d)$

$\mathsf{Sign}(sk, m \in \mathbb{Z}_N^*)$:
$\sigma = m^d \bmod N$

$\mathsf{Vrfy}(pk, m \in \mathbb{Z}_N^*, \sigma \in \mathbb{Z}_N^*)$:
Output 1 if and only if $m = \sigma^e \bmod N$

The correctness requirement is satisfied because

$$\sigma^e = (m^d)^e = m^{ed \bmod \varphi(N)} = m^1 = m \bmod N$$

Although the above scheme seems secure for real life scenarios it fails to satisfy the strict security definition that we gave in the previous lecture. Two attacks in the $\mathsf{Sig\text{-}Forge}$ experiment are the following:

Attack #1 (no queries):

According to the security experiment the adversary is given the public key $pk = (N, e)$. After getting it, he chooses a random $\sigma \in \mathbb{Z}_N^*$ and computes $m = \sigma^e \bmod N$ using the public key. He outputs the forgery (m, σ).

This is obviously a valid forgery since by construction $m = \sigma^e \bmod N$ and no queries were made. However one would argue that this attack seems "weird" or "unrealistic", because the adversary outputs a forgery on a message m that he cannot control. It is a random message since σ is random.

Attack #2 (stronger):

Suppose the adversary wants to forge on a specific message $m \in \mathbb{Z}_N^*$. He picks a random $m_1 \in \mathbb{Z}_N^*$ and sets $m_2 = m/m_1 \bmod N$ [8]. Then he makes two queries to the signing oracle on the messages m_1 and m_2.

[8] By m/m_1 we mean the multiplication of m to the inverse of m_1 modulo N; NOT integer division.

According to the experiment and the scheme, he gets back:

$$\sigma_1 = m_1^d \bmod N \text{ and } \sigma_2 = m_2^d = (m/m_1)^d \bmod N$$

Then he outputs the forgery $(m, \sigma = \sigma_1 \cdot \sigma_2 \bmod N)$. This is a valid forgery, because

1. $\sigma^e = (\sigma_1 \cdot \sigma_2)^e = (m_1^d \cdot (m/m_1)^d)^e = m^{de} = m^{de \bmod \varphi(N)} = m \bmod N$
2. m is different than m_1 or m_2

Although one can still argue that a real life adversary can not request for signatures on random messages, such as m_1 and m_2, we can "fix" the textbook scheme by introducing a hash function in the construction. The construction is shown in the following section.

60 Hashed RSA

The modification is that we add a suitable hash function $H : \{0,1\}^* \to \mathbb{Z}_N^*$ to the description of the scheme. The algorithms are the following:

$\mathsf{Gen}(1^n)$:
1. Run $\mathsf{GenRSA}(1^n) \to (N, e, d)$
2. Output $pk = (N, e, H)$ and $sk = (N, d, H)$

$\mathsf{Sign}(sk, m \in \{0,1\}^*)$:
$\sigma = H(m)^d \bmod N$

$\mathsf{Vrfy}(pk, m \in \{0,1\}^*, \sigma \in \mathbb{Z}_N^*)$:
Output 1 if and only if $H(m) = \sigma^e \bmod N$

Let's see why the previous attacks don't seem to work on the modified scheme and at the same time deduce some minimum requirements for H.

For the attack # 1, the adversary can still pick a random $\sigma \in \mathbb{Z}_N^*$ and compute an $\tilde{m} = \sigma^e \bmod N$. However now the forged message should not be $\tilde{m} \in \mathbb{Z}_N^*$, but a message $m \in \{0,1\}^*$ such that $H(m) = \tilde{m}$. Therefore the adversary has to find a way to invert H on $\tilde{m}$ in order to output (m, σ). So if the function H is not efficiently invertible this attack seems hard to work.

For the attack # 2, suppose he wants to forge on $m \in \{0,1\}^*$. He picks a random $m_1 \in \{0,1\}^*$ as before and queries the signing oracle for a signature on it. However now he can not query for m_2 because $m_2 = H^{-1}(H(m)/H(m_1) \bmod N)$: he again has to invert H. In general for this attack to work he should find m, m_1, m_2 such that $H(m) = H(m_1) \cdot H(m_2) \bmod N$. This seems hard if H is hard to invert.

Therefore, we can deduce that H should be at least hard to invert. Also, it is easy to see that it should be collision-resistant, since if collisions can be found easily then an adversary can find two different messages that have the exact same signature. Until now, no function H has been found such that the Hashed RSA scheme is existentially unforgeable under an adaptive chosen-message attack (the Sig-Forge game).

The scheme can be proved secure in this notion under the RSA assumption, when H is treated as a "Random Oracle" (see Chapter 13). However in all actual implementations H is replaced by a deterministic function; thus no strict security proof is known.