# Notes for G22.3025-001[*]
# (For Class Use Only)

[*]Last revised on: 2009-09-07 12:07:58 -0400 (Mon, 07 Sep 2009)

# 1 Introduction

The course description is posted in: http://www.cs.nyu.edu/courses/fall09/G22.3205-001/pages/00ReadMe_20090904.html. This file will be updated as needed, with revision date as part of the name (the last eight digits) updated.

I have written these notes to complement the textbook. Their table of contents will serve as a preliminary syllabus. I may modify slightly any part of the notes before presenting it in class.

I will put these notes on the class web, and I will mail to the class list the location. I do not want to put it in a publicly searchable place as it is not in a self-contained, finished form—it is being produced for this class specifically to complement the textbook. Whenever possible, I will rely on the book so to save time during presentation, although this will be more substantial later in the course.

I will also assign reading material for the book, generally material that is relatively simple so we do not need to cover it in class, or complementing what I cover in class.

**Note** The table of contents is at the end, it will be modified as the notes are modified. I do not expect considerable modifications, more like rephrasing and correcting typos, etc. I may also put additional material, as it seems useful.

## 1.1 Presentation of the material

It will be as informal as possible, sometimes "not quite right." This means more that it will not be wrong but "mathematically or otherwise not complete." But we will cover some aspects of relevant mathematics, assuming practically nothing beyond high-school mathematics.

We will not discuss the state of art as it existed in "earlier times," unless it is helpful to do so in order to understand what is done now (and in the near future; nobody can predict distant future anyway).

An important goal for us is to understand the intuition and *those parts of cryptography that are relevant now and in the near future* for building security systems. And they are actually quite simple, once you strip away the black magic.

The most important for us are the following fundamental cryptographic tools (though we will talk about others too):

1. Symmetric encryption (e.g., DES or AES)

2. Public/private encryption (e.g., RSA)
3. One way hash, or fingerprint (e.g., MD5)

From these, we will build (or discuss how to do it) various important and/or amusing applications, including

1. encrypting messages to others
2. encrypting files on your disk
3. signing messages with unforgeable signatures
4. SSL
5. digital cash
6. . . .

# 2 Setting

## 2.1 An ideal channel

We first discuss an ideal channel (or a pretty good one!). Alice wants to send messages to Bob. Look at Fig. 1.

Alice sends messages
Ideal Channel
Bob receives messages

Figure 1: Ideal Channel

We make some assumptions:

1. The channel is "physically" unbreakable: only Alice can put messages in and only Bob can take them out
2. The channel is opaque: nobody can see what is in transit
3. Messages are delivered, and in the order sent
4. There is a camera videotaping with timestamps everything Bob sees but only Bob can release parts of the videotape as he wants to (without making any changes to the tape)

Some of the useful properties we have:

1. *Privacy*: Nobody can see the messages unless Alice or Bob show them.

   In fact, it is not useful for Alice to show messages as she has no proof (in our setting) that she is going to later send them or that she has sent them. Unless of course she has an acknowledgment from Bob—this can be arranged but let's not talk about it now.
2. *Authentication*: Nobody other than Alice can put a message into the channel, so Bob knows that a message received must have been sent by Alice.
3. *Integrity*: Nobody can change the message in transit, so Bob knows that the message has not been garbled or modified by anybody else in transit.
4. *Non repudiation* (by Alice): Bob can prove that he got the message from Alice, if he wants to. He just shows an appropriate part of the videotape.

In reality, we have problems implementing such an ideal channel, as we assume that messages are sent using public networks and anybody can see them. We will work on how to do this. We will focus more on some issues more than on others—the ones that go beyond standard networking protocols, such as TCP `http://en.wikipedia.org/wiki/Transmission_Control_Protocol`. So major mechanisms will be encryption/decryption and digital signing.

## 2.2 Our (main) actors

We do not have (yet) and ideal channel. So difficulties pop up.

During various discussions and derivations we will have various *actors*, that is players or agents with various roles:[1]

1. Alice: This is an honest actor, unless we say otherwise. Sometimes she just works without anybody else, e.g., when she wants to encrypt (we use the term intuitively for now) some information on her hard disk.
2. Bob: This is an honest actor, unless we say otherwise. He appears when there is some communication going on between two (or among more than two) actors.
3. Carol: This is an honest actor, unless we say otherwise.
4. Doug: This is an honest actor, unless we say otherwise.
5. Eve: An adversary; she is an *eavesdropper*. Can see messages in transit. But she cannot interfere with the traffic Her goal is to learn something about the contents of the messages. (She can perform reasonably long computations on them using reasonable amount of storage trying to figure out what they mean—not something that will take the life of the universe or huge number of bits, but maybe a few years on a very powerful parallel computer and exabytes of storage is OK; more about this later).
6. Mallory: An adversary; he is *malicious*. Can do all that Eve can do, but can also inject messages, remove messages, garble messages, etc. His goal is to confuse the honest actors, and/or take advantage of them (by perhaps stealing money from them).
7. Peggy: Peggy is an honest actor, she needs to *prove* that she knows some secret.

[1]See also `http://en.wikipedia.org/wiki/Alice_and_Bob`.

8. Trent: A *trusted* authority. He is like a notary public. Whatever he does is to be believed (of course, once we know that he did it and we assume that nobody else has stolen his notary seal or forged his signature).

9. Victor: A *verifier*. He is also trusted and he verifies that what somebody else did indeed satisfied what needs to be satisfied.

Our key goal is: protecting the communication from Alice to Bob (and from Bob to Alice too, if we want to).

We do not focus on issues such as: cutting the channel, denial of service, worms, span, viruses, bugs in O.S., keystroke logging (beware of reading email in Internet cafes!), etc.

We worry about problems that can be prevented using cryptography. We want to recreate some version of an ideal channel, with the goal that not a single bit leaks out.

## 2.3 Three fundamental building blocks

We will rely on the following building blocks

1. Symmetric encryption

2. Public/private encryption

3. Fingerprinting (One-way hash)

Items 1 and 3 do not really have as solid mathematics behind them as one may want. Item 2 is better in this sense.[2] Everything can be done using item 2, but there are reasons why all three are used—we discuss later.[3]

We will understand these blocks in detail later, right now we will learn enough to know what they (roughly) mean and how they can be used.

## 2.4 Key ideas

1. The ideal channel

2. The building blocks (to be understood soon).

[2]Or perhaps worse, as we will discuss. Basically the nice solid mathematics could lead to efficient algorithms for breaking the encryption system.

[3]Mainly because, if we only use item 2, there are some technically unpleasant things to do and it would be much more inefficient.

## 2.5 Please read

Forouzan Sections 1.1–1.4.

# 3 Some concepts

## 3.1 Example: Caesar's cipher

We will use this example of a 2000-year old "privacy method," to explain some basic concepts that we will use.

Of course, Caesar wrote in Latin as it was written when he was alive http://en.wikipedia.org/wiki/Latin_alphabet, and therefore he used 23 letters (and in capitals only): A, B, C, D, E, F, G, H, I, K, L, M, N, O, P, Q, R, S, T, V, X, Y, Z.[4] (Note that sometimes we pretend that we write using a "classical" Latin alphabet. Look at the sign above a door on Greene Street between West Fourth Street and Washington Place. You will see "VNIVERSITY" there.)

Caesar wrote messages (strings) using 23 letters (alphabet of 23 letters). He (and others at that time) did not generally use spaces and punctuation marks, so let's assume that spaces are not written at all.[5] (Or if you like spaces are left unencrypted, which is an extremely bad idea, as Eve can immediately see which pieces of the encrypted message are the individual words.)

He used the following simple encryption method.[6] Every letter was replaced by a letter 3 positions to the right, circularly. Therefore: $A \to D$, $B \to E$, $\dots$, $V \to Z$, $X \to A$, $\dots$

To encrypt a string do this for each letter (independently). So $VENI \to ZHQM$

The corresponding decryption method was "going in reverse." That is: $A \to X$, $B \to Y$, $\dots$, $Z \to V$, $\dots$ We do this for each letter separately.

Of course, one could you use a parameter other than 3 for rotation, in fact any number in $\{0, \dots, 22\}$. So, if say, 4 is chosen, then, in encryption: $A \to E$ and in decryption $A \to V$, etc.

Let us now generalize. We can assume an existence of two *machines* for using a Caesar type method: an *encrypting machine* $E$ and a *decrypting machine* $D$. These machines could, of course, be implemented in software.

For $E$ we can choose a parameter $e$ (how many positions to rotate (more precisely later), technically called *an encryption key*. For $D$ we can choose a parameter $d$ (how many positions to rotate (more precisely later), technically called *a decryption key*.

[4]Something about Julius Caesar that we do not have to know: http://en.wikipedia.org/wiki/Julius_Caesar.

[5]See e.g., http://en.wikipedia.org/wiki/Arch_of_Titus.

[6]You may read about it also at: http://en.wikipedia.org/wiki/Caesar_cipher.

Let's consider a hardware implementation of $E$. Look at Fig. 2.

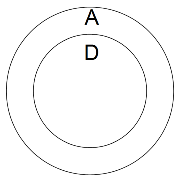


Figure 2: Hardware implementation of Caesar's encryption machine with parameter 3. So the letter "D" is rotated to be opposite to "A".

There are two concentric rotors, each having all the 23 letters written around. To choose parameter $e = 3$, just rotate the inner rotor to the left so that the the first letter $A$ is aligned with the fourth letter $D$. Then you can read out what to encrypt into what. So the description of using the machine is:

1. Choose $e$
2. Rotate inner rotor by $e$ positions to the left ("left" is part of the definition of the machine $E$ and not of choice of the key $e$)
3. Use alignment of letters for encryption

The above looks like a trivial and unnecessary formalization but it is useful for clarifying definitions.

What about a hardware implementation of $D$? We have to figure what $D$ and $d$ should be. We have two obvious choices.

We could choose two variants:

1. $D = E$ and $d = 23 - e$.
2. $D$ to be similar to $E$, but with rotation going right; then $d = e$.

So, after we choose one variant of $D$, we have two machines (one of encrypting and one for decrypting),[7] and we can schematically describe the situation as depicted in Fig. 3 and Fig. 4. In both, $m$ is a *message* and $c$ is a *cipher*, that is the encrypted version of a message.

We can think of these machines as implementing functions $E(e, x)$ and $D(d, x)$, where $x$ is a letter.

[7]We could have just used one machine, but it is better to talk about two machines.

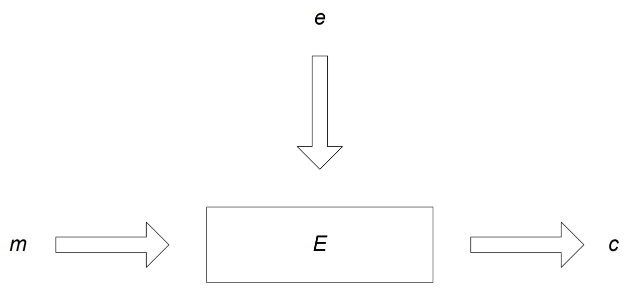


Figure 3: Encrypting Machine

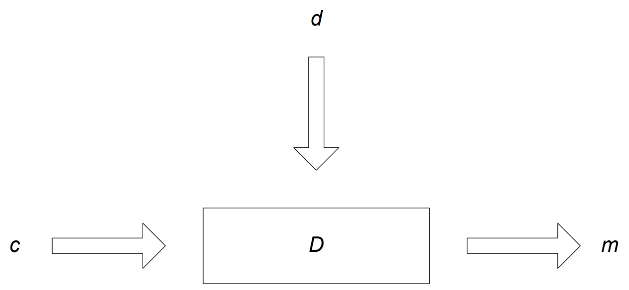


Figure 4: Decrypting Machine

The important property is that if $e$ and $d$ are related as to be "inverses" of each other as implied by our choices of $E$ and $D$, then for any letter $x$, $D(d, E(e, x)) = x$.

So, if Alice and Bob want to communicate using Caesar's cipher, they have both $E$ and $D$ and they agree on an *encryption key* $e \in \{0, \ldots, 22\}$. From this they can easily compute the *decryption key* $d$.

In fact if Alice sends messages to Bob, and Bob does not send messages to Alice, Alice needs to know only $E$ and $e$ and Bob needs to know only $D$ and $d$. This will be important soon, but not right now.

Alice, to encrypt some message $m$, computes $E(e, m)$. For Caesar's cipher, if $m = m_1 \ldots m_n$, then $E(e, m) = E(e, m_1) \ldots E(e, m_n)$. If $E(e, m_i) = c_i$, then the encryption of the message, that is the *cipher* is $c = c_1 \ldots c_n$.

When Bob gets the string $c$, he interprets it as cipher and computes $D(d, c_1) \ldots D(d, c_n)$. But $D(d, c_i) = m_i$, so he gets $m_1 \ldots m_n = m$, the original message.

Let's slightly generalize. We have

1. $M$: set of all messages, here all strings of letters, that is $\{A, \ldots, Z\}^*$

2. $C$: set of all ciphers (universe of encrypted messages), here all strings of letters, that is $\{A, \dots, Z\}^*$
3. $E$: an encrypting machine, as discussed above
4. $D$: a decrypting machine, as discussed above (let's look at the first variant—see how $d$ is chosen next)
5. $K$: a *key space*, a set of pairs of the form (encrypting key, corresponding decrypting key), here $(0, 0), (1, 22), \dots, (22, 1)$.[8]

Then, once $(e, d)$ is chosen from $K$ ($e$ explicitly, $d$ perhaps computed from $e$), we have that for every $m \in M$: $D(d, E(e, m)) = m$.

## 3.2 A general setting

A general cryptosystem consists of a tuple of 5 elements: $(M, C, K, E, D)$ with the following properties:

1. $M$: set of all messages
2. $C$: set of all ciphers (universe of encrypted messages)
3. $K$: a *key space*, a set of pairs of the form $(e, d)$, associating a decrypting key with an encrypting key. In practice it generally makes sense for this to be a one-to-one function, that is for each encrypting key there is exactly one decrypting key and vice versa.

   Let $K_{\mathrm{E}}$ be the set of all the encrypting keys and let $K_{\mathrm{D}}$ be the set of all the decrypting keys; we get them from the set $K$ above.
4. $E$: an encrypting algorithm/machine, $E\colon K_{\mathrm{E}} \times M \to C$. That is, given an encrypting key $e$ and a message $m$, we get the corresponding cipher: $c = E(e, m)$.
5. $D$: a decrypting algorithm/machine, $D\colon K_{\mathrm{D}} \times C \to M$. That is, given a decrypting key $d$ and a cipher $c$, we get the corresponding message: $m = D(d, c)$.
6. For any $m \in M$ and $(e, d) \in K$: $D(d, E(e, m)) = m$

**Note** We did not need to explicitly define $K_{\mathrm{E}}$ and $K_{\mathrm{D}}$, defining just $K$ was sufficient for our purpose.

[8] $(0, 0)$ is also $(0, 23)$.

**Remark** For simplicity, we may assume for a while (while in cryptographic domain) that $M = C$. This is definitely *not true* in general once cryptography is used in actual protocols, but not conceptually important for now. Why would $M \neq C$? A simple reason (but there are deeper ones too) you may want messages to have some specific structure but the encrypted versions of them may have different structures.
Here is a real message:

```
Hello, world!
```

And here is its cipher:

```
-----BEGIN PGP MESSAGE-----
Version: GnuPG v1.4.9 (MingW32)
Comment: http://getfiregpg.org

hQIOA+Z/z6+DggDEEAf/Tr5A3LBrh2iZD6Gvjz+xJlaIDh5NFA/OghSb/5HSRJ5C
jr7TyrihdQTfbuAa4RmAhRN14V47ZjuYpe/JQKe0rbQMP6pgTDxyrhgjelzvUdh6
oXj6kLXFdTdBf5JCCOhwTw0WvoQT/Pb1RTiD6USZFzsMKRXugotiL8TnYMB32HaW
o7stn9krE53uBibihP5yqh0tJseJNAUL0RPjQtjBqk+DGPyGPUdnSf5C33yIamXc
0XiIIa33IM68FcrM4Y9BlFq1dIk0YADodgQOR/fd2LsQA1r3Mvmimy3iVTiWqRgD
3NZdfaKgUIAfD2vBGQ011idkMZ3JcUK3nUDBqP+K8QgAssn6Y3zjkMLsN7gzR3BV
Nuzy2F+YHqg8jcOFlaqSUIjbKrZBSEEWSLt5UrzaSI7Ml3sLHoGHsnBGhiK5lvKZ
eAVk0c1mcOil8j3s7Pg2PjyYU9DxOdwEATF06aU6Ew86UGWc7TaIASINi6hrYJ/+
ZfrF9PCCwoGd8xov960v5Kogqdw8aS5e99rp0gNgJpa/EXVNqnCLDUJL4mPgQ0of
MTv7ku8TOyOK7vylra7qfdan7obWvhECgyxZTUJhQu57Dc/AZ0siKrej9scueqd0
Wl68DJAmYKOX2nypO4JfEOUi0fCx0hZvvFDAqRjQ8l2CpemA2uK1cvGvtrdFsxqy
jtJjARV5lgnbiPL8G0meoJRnNVwoiOIDXE8LLmZg/K8GYnoVwfapRlhKD6pZVLYx
afy5t+eT5kxS2h97+93PVlP557XfadH7Oj7pz2C3HxcAjLXmSTk6jQn6hWus1ByO
sKeWkMzt
=WaIh
-----END PGP MESSAGE-----
```

In this specific system all ciphers have to have a well-defined structure with a specifice header, etc. Therefore, the universe of all ciphers in this system cannot consist of all strings.

**Remark** Also, we will assume that once a pair $(e, d)$ is chosen $E(e, \cdot)$ and $D(d, \cdot)$ are one-to-one functions and inverse of each other (see Fig. 5). Therefore, for instance: $E(e, (D(d, x)) = x$. This is all very natural in our setting,

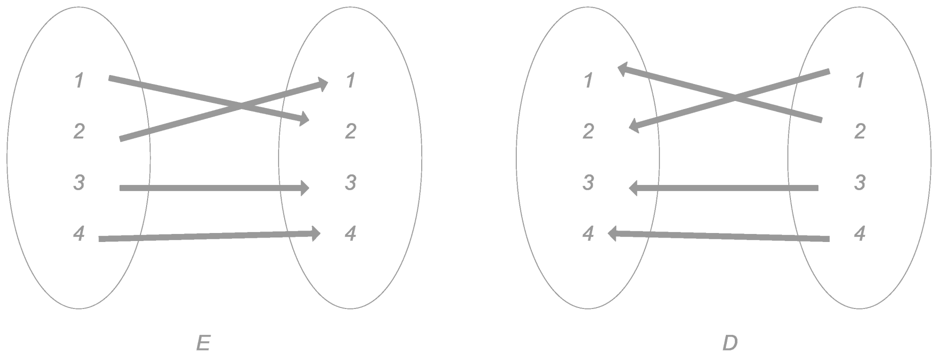


Figure 5: Example of functions $E$ and $D$ for some specific choice of $(e, d)$. $M = C = \{1, 2, 3, 4\}$.

but needs to be modified in a non-consequential way for actual protocols, as we will do later.

There are two types of cryptosystems:

1. Symmetric systems such as DES and AES, relying on *confusion* and *diffusion*
2. Public/private systems, such as RSA and El Gamal, relying respectively on the presumed computational difficulty (resources required) of factoring a composite integer into prime factors [9] and of computing discrete log.[10]

We will explain the difference between the two types shortly.

## 3.3 Kerckhoffs' main desideratum

In 1883 Kerckhoffs http://en.wikipedia.org/wiki/Auguste_Kerckhoffs, not Kirchhoff http://en.wikipedia.org/wiki/Gustav_Kirchhoff, formulated some requirements for cryptographic systems, one of them (the second principle) we can reinterpret as follows

> When Alice and Bob communicate, using a cryptographic system $(M, C, K, E, D)$, it should be *completely public*, that is $M$, $C$, $K$, $E$, $D$ should be known to everybody (or more weakly stated: cannot be assumed to be secret)

[9] E.g., writing 50 as $2 \times 5^2$.

[10] You are not expected to know now what discrete log is.

For Caesar's systems we have specified the above completely. What about the pair $(e, d)$ that Alice and Bob use to communicate? We will see soon.

Kerckhoffs' motivation was that for cryptographic systems at his time an encryption/decryption machine was a heavy machine that could fall into enemy's hands (just like the Enigma machine during WW II http://en.wikipedia.org/wiki/Enigma_machine), so we cannot rely on its being secret.[11]

Modern motivation: Vendors who sell us cryptographic systems should not sell opaque hardware/software (systems should be open source) because:

1. Vendors could have buggy systems and nobody would know about this

2. Vendors could on purpose build some holes/trapdoors in the system so that they can in future decrypt encrypted messages of the users of the system without getting the users' permission or even them knowing about this

## 3.4 Symmetric systems and public/private systems

Let us now return to the key space $K$. Alice and Bob have chosen a cryptosystems using which Alice will send messages to Bob: Alice encrypts and Bob decrypts. Alice *must* know $e$ and Bob must know $d$.

If encrypted messages could be visible to anybody while in transit—which is the assumption we of course make—then $d$ must be kept secret, restricted only to people who are permitted to decrypt messages sent to Bob. (Generally only Bob.)

*If the decryption key $d$ can be easily computed from the encryption key $e$ (such as in Caesar's cipher)*[12] *than $e$ must be kept secret too.*

A cryptosystem is called *symmetric* if and only if $d$ is easily computable from $e$.

But if $d$ is not easily computable from $e$, then $e$ may as well be made public (put on a Web site as in http://www.cs.nyu.edu/kedem/0xC6540406.asc.

Then *anybody* can send encrypted messages (ciphers) to Bob and only he can decrypt them, as he is the only one who knows the corresponding $d$.

For convenience of explanation, a message is put in a box. Originally it is unlocked, this is depicted in Fig. 6. Then, whoever has $e$ (everybody) can rotate the arrow to the right, thus locking the box, this is depicted in

[11]One machine was used for both decryption and encryption, which we saw was possible in the case of Caesar's cipher.

[12]$d = 23 - e$, for every $e$.

Fig. 7. Then, whoever has $d$ (Bob only) can rotate the arrow to the left, thus unlocking the box, this is depicted in Fig. 8.

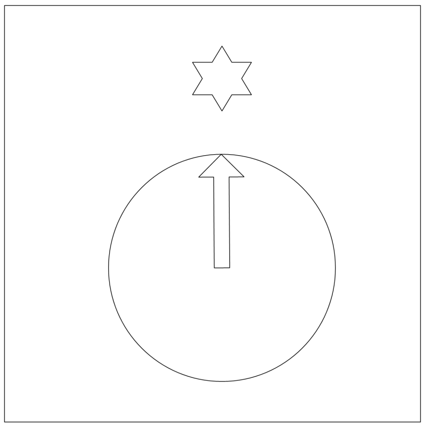


Figure 6: The box before locking

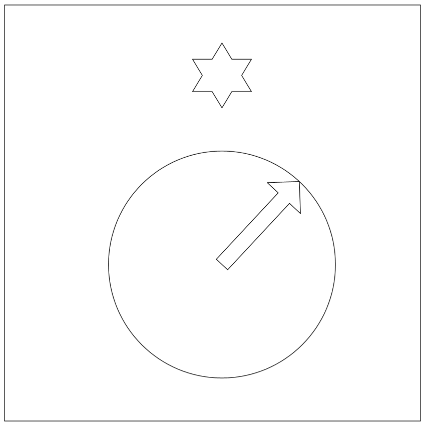


Figure 7: The box after locking

Such a cryptosystem is called *public/private*.

## 3.5 One-time pad

See also Forouzan, Section 5.2, p. 149. Alice and Bob will want to communicate in the future: Actually Alice will want to send a string of $\leq n$ bits to Bob in a secure way. They will use a one-time pad `http://en.wikipedia.org/wiki/One-time_pad`.

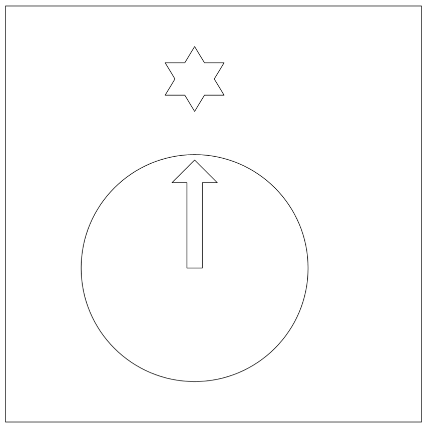


Figure 8: The box after unlocking

Alice and Bob agree on a encryption key, a *random* (not pseudorandom!) string of $n$ bits $e = e_1 \ldots e_n$. The decryption key $d$ will be the same as $e$, so $e = d$. Of course we have a symmetric system here and both keys must be kept secret. Alice and Bob have to find a secure way to agree on this key. Possibly, Alice generates it and arranges for secure delivery to Bob (through a trusted courier, perhaps).

We will use the symbol $\oplus$ to stand for exclusive OR, which is also addition modulo 2 in the set $\{0, 1\}$. (Fancy name for the latter: $\mathbf{GF}(2)$)

Alice wants to send a message $m = m_1 \ldots$ of length $\leq n$ to Bob. For simplicity, we will assume that it is of length $n$ exactly. She encrypts bit by bit:

$$E(e, m) = e_1 \oplus m_1 \; e_2 \oplus m_2 \ldots e_n \oplus m_n = c_1 c_2 \ldots c_n = c$$

That is, $c_i = e_i \oplus m_i$.

Bob decrypts:

$$D(d, m) = d_1 \oplus c_1 \; d_2 \oplus c_2 \ldots d_n \oplus c_n$$

But what does he get? Recalling that $e_i = d_i$, we have:

$$c_i \oplus d_i = c_i \oplus e_i = (m_i \oplus e_i) \oplus e_i = m_i \oplus (e_i \oplus e_i) = m_i \oplus 0 = m_i$$

Therefore he gets $m$ back.

Following Shannon, we will prove later that this system is *provably unbreakable.* In fact, it is *the only system that is known to be provably* *unbreakable*. Slightly more formally, we can say that if Eve knows that the one-time pad system was used, that is she knows:

1. $E$ and $D$ are one-time pad encrypting and decrypting machines, and
2. she knows *all* of $c$

she cannot learn *anything* about $m$. (This is still very imprecise statement). Intuition behind this:

$$\forall c \, \forall m \, \exists e \, [E(e,m) = c]$$

So any $c$ could have come from any $m$ under appropriate $e$, in fact $e = m \oplus c$.

What if Alice uses the same $e$ more than once. Potentially a problem. We will use the notation $\oplus$ for strings in an obvious way, when applied to two strings of equal length this will mean doing $\oplus$ bit-wise.

Let's consider an example. One time pad $abc$. Two messages $m_1m_2m_3$ and $n_1n_2n_3$. Assume that it happens that $abc \oplus m_1m_2m_3 = \alpha\beta\gamma$ and $abc \oplus n_1n_2n_3 = \delta\beta\epsilon$. Then Eve knows that $m_2 = n_2$ and other bits are different in the two messages. And this may be useful.

Let's look at another thing that Eve can do. Eve gets $c^{(1)} = e \oplus m^{(1)}$ and $c^{(2)} = e \oplus m^{(2)}$. She can compute and computes $f = c^{(1)} \oplus c^{(2)}$. Now $f$ has 0's in positions where $c^{(1)}$ is equal to $c^{(2)}$. But these are also positions in which $m^{(1)}$ is equal to $m^{(2)}$. Assume that Alice sends email from NYU and $f = \ldots 10 \ldots 01 \ldots$ and the length of the substring of 0's is the length of the string " New York University ". She guess this is indeed the string that appears in the same position in both the messages (perhaps it is a header that is used for some messages. Then she can guess some bits of $e$. And if $e$ is used again, some parts of subsequent messages can be recovered from their ciphers.

In general pieces can leak, for instance as the letter "e" is the most common letter in English (but look at http://en.wikipedia.org/wiki/A_Void).[13] This can be used as follows. "Many" of the places in which two unrelated messages have the same letter are more likely to be the letter "e"

[13] A passage thanks to http://www-control.eng.cam.ac.uk/hu/Perec.html: Noon rings out. A wasp, making an ominous sound, a sound akin to a klaxon or a tocsin, flits about. Augustus, who has had a bad night, sits up blinking and purblind. Oh what was that word (is his thought) that ran through my brain all night, that idiotic word that, hard as I'd try to pun it down, was always just an inch or two out of my grasp - fowl or foul or Vow or Voyal? - a word which, by association, brought into play an incongruous mass and magma of nouns, idioms, slogans and sayings, a confusing, amorphous outpouring

than another letter. This again can provide useful information about the messages and $e$. Perhaps some parts can be guessed.

We looked at the ultimate secure symmetric cryptosystem. To prove it is really secure (and actually define some properties more precisely) we will need to know some probability (which we will learn, nothing is assumed). It is still supposedly used for ultrasecret communications.

But this system is, of course, not practical for normal use. (How do you distribute the keys? Keys also have to be as long as the messages.) So how do symmetric systems work in practice?

We will cover this later in the course. But here is a rough idea. A fixed-size key of length $e$, say 256 bits is chosen randomly (one hopes randomly not pseudorandomly). An encrypting machine (chosen earlier) takes a message $m$ and creates a very different looking $c$ from it guided by $e$. A fake example: If the first bit of $e$ is 1 than exchange in $m$ the first bit with the second bit, the third with the fourth, etc, otherwise don't; take the second bit of $e$ and do XOR with every bit of $m$.

Ideally every bit of $c$ depends on every bit of $m$; changing one bit of $m$ changes approximately half the bits of $c$ in random-looking positions (not actually random, $E$ is deterministic. So Eve has to work very hard to guess $d$. (Recall that $e$ and $d$ are closely related in a publicly known way, so this is as hard as guessing $e$.)

## 3.6 Key ideas

1. Definition of a cryptosystem

2. Machines, $E$ and $D$ known to all

3. Knowledge of $d$ (decryption key) always controlled; knowledge of $e$ (encryption key) sometimes controlled.

4. In symmetric systems, it is easy to determine $d$ from $e$, therefore knowledge of $e$ is controlled too

which I sought in vain to control or turn off but which wound around my mind a whirlwind of a cord, a whiplash of a cord, a cord that would split again and again, would knit again and again, of words without communication or any possibility of combination, words without pronunciation, signification or transcription but out of which, notwithstanding, was brought forth a flux, a continuous, compact and lucid flow: an intuition, a vacillating frisson of illumination as if caught in a flash of lightning or in a mist abruptly rising to unshroud an obvious sign - but a sign, alas, that would last an instant only to vanish for good.

5. In public/private systems, it is not easy to determine $d$ from $e$, therefore knowledge of $e$ is not controlled
6. The one-time pad symmetric system

## 3.7 Please read

Forouzan, Section 3.1, pages 56–57.

# 4 Fundamental protocols

## 4.1 How do I know I am talking to Amazon?

This is a good question to motivate some important developments, as well as being important by itself.

We are now ready to sketch the main idea behind amazon.com authenticating itself to Bob. It is only a very rough description and not complete but it covers the essence of what is going on—we will give a more complete description later. Everybody uses some public/private cryptosystem $(M, C, K, E, D)$. For simplicity, assume that $M = C$. Perhaps these are all non-negative integers or all bit strings.

amazon.com (call it Alice from now on) has a public/private key pair $(e, d)$. Everybody (including Bob) knows $e$ but only Alice knows $d$. So if an entity proves it knows $d$, then this entity must be Alice. Alice does this by showing Bob that she can decrypt strings using Alice's private key $d$; and without knowing that key this is not possible. [14]

### 4.1.1 Simple protocol: good idea, which has some problems

Bob want to *challenge* Alice to prove she is Alice. We will refer to Bob as the *challenger* and to Alice as the *challengee*.[15]

So here is a candidate protocol for this (see Fig. 9):

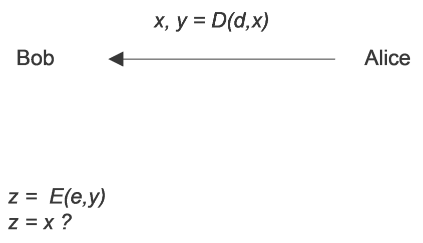


Figure 9: Alice authenticates herself to Bob. Source of $x$ unspecified.

1. Alice has a challenge $x$ [16]

2. Alice "decrypts"[17] $x$ getting $D(d, x) = y$

[14] We could have used Peggy (prover) for Alice and Victor (verifier) for Bob.
[15] "Challengee" is not really a word in English, but very useful for us.
[16] We discuss soon where $x$ could come from.
[17] There was really nothing to decrypt, but the decryption operation was applied.

3. Alice sends $x$ and $y$ to Bob
4. Bob encrypts $y$ getting $E(e,y) = E(e,D(d,x)) = z$
5. Bob checks whether $z = x$. If yes, he believes he is talking to Alice, since only Alice knows how to compute $D(d,x)$ given $x$ as she is the only one who knows $d$.

Where did $x$ come from? Let's consider some choices.

### 4.1.2 Bob picks the challenge

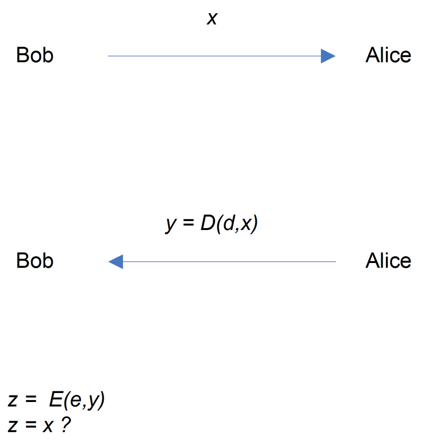


Figure 10: Alice authenticates herself to Bob. Bob picks $x$.

Bob picks $x$. The protocol of Section 4.1.1 is modified as follows (see Fig. 10):

1. Bob picks a *challenge* $x$
2. Bob sends $x$ to Alice
3. Alice "decrypts" $x$ getting $D(d,x) = y$
4. Alice sends $y$ to Bob [18]
5. Bob encrypts $y$ getting $E(e,y) = E(e,D(d,x)) = z$

[18]Bob already knows $x$.

6. Bob checks whether $z = x$. If yes, he believes he is talking to Alice, since only Alice knows how to compute $D(d, x)$ given $x$, as she is the only one who knows $d$.

**A Problem:** Some time in the past, Carol wanted to send a confidential message $u$ to Alice. So, she encrypted it with Alice's public key $e$ and sent her $v = E(e, u)$. Mallory saw $v$ in transit and would like to know what $u$ was.

Mallory tells Alice that he wants to buy a book from her, so he needs to know that he is talking to her.

He follows the protocol that Bob follows.

1. Mallory picks a challenge $v$ (the specific $v$ above)
2. Mallory sends $v$ to Alice
3. Alice decrypts $v$ getting $D(d, v) = D(d, E(e, u)) = u$
4. Alice sends $u$ to Mallory

Mallory does not modify it and has the secret message $u$

**Another problem:** Sometime in the past Bob authenticated himself to Alice using challenge $x$. Mallory saw both $x$ and $y$ in transit and remembers them.[19]

Carol want to buy a book from Alice. Carol sends Alice a challenge, which happens to be the "old" $x$.

Mallory intercepts this $x$ and send Carol the corresponding $y$. Carol believes she is talking to Alice.

**Moral:** The challenger cannot be permitted to pick the challenge.

### 4.1.3 Alice picks the challenge

We have seen that the challenger cannot be allowed to pick the challenge. What if the challengee does it?

Bob wants Alice to prove she is Alice. Here is a modification of the protocol in Section 4.1.1, which now specifies that Alice picks $x$ (see Fig. 11):

[19]Mallory has a lot of storage, more than other participants, who cannot remember all of the past.

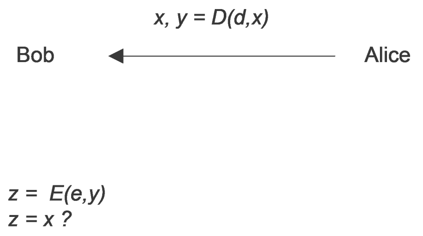


Figure 11: Alice authenticates herself to Bob. Alice picks $x$.

1. Alice picks a challenge $x$
2. Alice "decrypts" $x$ getting $D(d, x) = y$
3. Alice sends $x$ and $y$ to Bob
4. Bob encrypts $y$ getting $E(e, y) = E(e, D(d, x)) = z$
5. Bob checks whether $z = x$. If yes, he believes he is talking to Alice, since only Alice knows how to compute $D(d, x)$ given $x$ as she is the only one who knows $d$.

Mallory can see $x$ and $y$ while they are in transit. He can now pretend to be Alice.

Carol wants to buy a book from Alice and asks her to prove she is Alice. Mallory intercepts this request and replays to Carol $x$ and $y$ that Alice previously sent to Bob. So Carol believes she is talking to Alice.

**Moral:** The challengee cannot be permitted to pick the challenge.

**Trent ?** So perhaps Trent, whom everybody trust, should keep track of the complete history of the system and send an $x$ to Alice and Bob, when Bob wants to authenticate Alice.

This is:

1. Inefficient
2. Nonscalable
3. Requires some protocol so that Alice and Bob believe that $x$ came from Trent and not from Mallory.

So we need to think a little more. We will, in effect, "simulate" Trent, using a *one-way function.*

## 4.2 One-way functions

See Forouzan, Section 10.1, p. 297, Section 11.1, pp. 340–343. He uses somewhat different terminology. Also, I will need to make it "more correct," So I am going over this, without the figures, which are good: 11.1–11.6. See also, http://en.wikipedia.org/wiki/Cryptographic_hash_function.

**Definition 1.** Let $X$ and $Y$ be sets. Let $f\colon X \to Y$. Then $f$ is called a one-way-function iff:[20]

1. $f$ is *easy to compute.*

   We do not formally define "easy to compute", intuitively clear: efficient (fast, small space, etc.)

2. *For practically every* $y \in Y$ it is *computationally infeasible* to find *any* $x \in X$ s.t. $f(x) = y$.[21] Note, that in general there could be many such $x$'s as it is not assumed that $f$ is a one-to-one function. Frequently the term *Preimage Resistance* is used.

   We do not formally define "computationally infeasible". This intuitively means that no amount of reasonable time and reasonable amount of space (storage) would allow us to find such an $x$. The "practically every" is a little more trickier, we *cannot* say "for every", as we will discuss shortly.

3. For every $x_1 \in X$ it is *computationally infeasible* to find any $x_2$ s.t. $f(x_1) = f(x_2)$. This condition is called *weak collision resistance.* Frequently, the term *Second Preimage Resistance* is used.

4. It is *computationally infeasible* to find any pair $x_1$, $x_2$, $x_1 \neq x_2$, s.t. $f(x_1) = f(x_2)$. This condition is called *strong collision resistance.* Frequently, just the term *Collision Resistance* is used.

(This definition is not completely formal, but sufficient for our purposes.)

□

Let us now discuss why the following *cannot be true*:[22]

> *For every* $y \in Y$ it is *computationally infeasible* to find *any* $x \in X$ s.t. $f(x) = y$.

[20] "iff" stands for "if and only if"
[21] "s.t." stands for "such that."
[22] This point is not always correctly discussed.

Very simple reason. Compute $f(0)$. You will get some string, say $\alpha = 0100110011000011100011011110010100010011000011000011111$. Remember this result for input 0. Then, if somebody asks you in the future to find $x$, such that $f(x) = \alpha$, you can just look up in your notes and answer: 0.

But you can do this trick for only a "small" number of $x$'s, due to lack of time to precompute "many" answers and even worse, due to the lack of space to store "many" answers.

Forouzan gives examples of functions that are not one-way.

## 4.3 Fingerprint, or one-way hash

Let $X$ be the set of all binary strings and let $Y$ be the set of all binary strings of certain length, let us say 128 bits (160 bits or 256 bits are also common). Then, a one-way function as above is called a *fingerprint* or a *one-way hash*, for the origin of the term, see `http://en.wikipedia.org/wiki/Hash_function#Origins_of_the_term`. Example of such a function is MD5. Some software: `http://www.softpedia.com/get/System/System-Miscellaneous/MD5-Fingerprint.shtml`. Another popular one is SHA-1 `http://en.wikipedia.org/wiki/SHA-1`. Sometimes we will refer to such a function using $h$. For such schematic "hashing machine," look at Fig. 12.

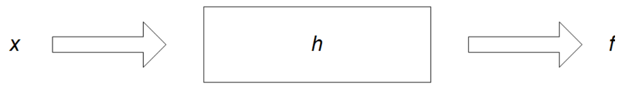


Figure 12: One-way hashing machine. $f$ refers to the resulting fingerprint. $x$ is of any (finite) lengths and $f$ is of fixed length.

How could we obtain such a function $h$? We could (what is not done in practice—we will talk about practice later in the course) have a very "mixing up functions" that takes $x$ (assume relatively long), mixes up its bits in some way, performs operations on them and finally takes 128 of them out in some way. It may be difficult (and we assume it is computationally impossible), to "reverse it", that is to find any $x$ that produces some given $y$ (for practically all $y$'s).

There are several different choices for $h$, none of them secret and everybody knows how to compute them. Let us assume that one is picked out for the following discussion. We will look at some examples how this is used.

### 4.3.1 Message integrity

Alice sends a (relatively) very long $m$ to Bob. She also computes $h(m)$. Bob gets some message from Alice, say $m'$. He wants to verify that he got the original $m$, that is that $m' = m$. See how it is done in Forouzan, Section 11.3, pp. 352–353, Fig. 11.9. The "secure channel" could be a phone call.

Similarly, one can check that downloaded software has not been garbled in transmission, see `http://www.gimp.org/downloads/`, they use the term "MD5 sum" where we say "MD5 hash."

Also, if you look at my public key and want to verify that it is really mine, you can call me and I will tell you the fingerprint (hash) on the phone: `http://www.cs.nyu.edu/kedem/contact.html`.

### 4.3.2 Tossing coins over the phone

Alice and Bob want to decide who wins. Alice tosses a coin, which has H (heads) and T (tails) and Bob guesses. If Bob guesses correctly, he wins, otherwise Alice wins. Actually, it is the same as Alice's picking up H or T without any tossing and Bob guessing what she has picked. Instead of H and T, we will use bits 0 and 1.

Easy to do it "physically", we will do it over the phone. In fact, we could even do it by email.

We do it "half by example," but the general protocol is clear.

1. Alice picks a random bit string. If she picks H, she appends 0 to the string; if she picks T, she appends 1 to the string. Let the resulting string be $x$. Let us say she picked T, so the last bit is 1. She computes $h(x) = y$

2. Alice sends $y$ to Bob. At this point she is *committed*

3. Bob guesses 0 (that is H) or 1 (that is T), and tells Alice his guess. Let us say he guesses 0, so he loses

4. Alice wants to show Bob he lost.[23] She sends him $x$

5. Bob computes $h(x)$, sees that this is indeed $y$. He knows that Alice could not compute another "pre-image" for $y$, so her original choice must have been $x$. Therefore her original choice had 1 as the last bit and he admits losing

[23]If he wins, she concedes defeat.

#### 4.3.3 Password storing in the clear

This is not a complete solution (we will complete later). Forouzan, Section 14.2, p. 418, Fig. 14.2.[24]

### 4.4 Digital signatures

See also, Forouzan, Section 13.2, pp. 390–392 and the first part of `http://www.youdzone.com/signature.html`.

Let $(M, C, K, E, D)$ be a public/private cryptosystem. Alice has $(e, d)$. Everybody knows $e$, but only Alice knows $d$. Alice will be sending messages to Bob and signing them.

You will notice similarity to Section 4.1.

So when Alice sends $m$ to Bob, she also sends $s$, her signature on $m$. The signature may be detached from the message.

So she effectively sends two messages. It will be convenient for us, sometime, to denote the two messages that Alice sends as $x_1$ and $x_2$.

Let us attempt to produce a signature. Our first attempt is somewhat similar to the "Amazon authentication" scenario in Sec. 4.1. Look at Fig. 13.

Alice wants to sign some message $m$.

1. Alice computes $s = D(d, m)$. Let us use the notation $x_1 = m$, and $x_2 = s$.

2. Alice sends Bob $x_1$ saying this is the message and $x_2$ saying this is the signature.

3. Bob computes $z = E(e, x_2)$. If $z = x_1$ he accepts $x_2$ as Alice's signature on $x_1$

There are problems with this attempt, similar to what we discussed previously. We will just look at an example scenario.

Here is what Mallory can do. He takes some string $s$ and computes a string $m = E(e, s)$. He sets $x_1 = m$ and $x_2 = s$. He tells Bob that $x_1$ is a message and $x_2$ is Alice's signature on it.[25]

As before, Bob computes $y = E(e, x_2)$. If $y = x_1$ he accepts $x_2$ as Alice's signature on $x_1$. So let's actually compute it: $y = E(e, x_2) = E(e, s) = m = x_1$. And he accepts this! (We need to look carefully of what has happened,

[24]Of course, this implies that if you lost your password, the server cannot tell you what it is, and generally sends you a temporary password, or gives you a one-time URL to create a password, or some variant of these.

[25]$x_1$ is a "raw," unencrypted message.

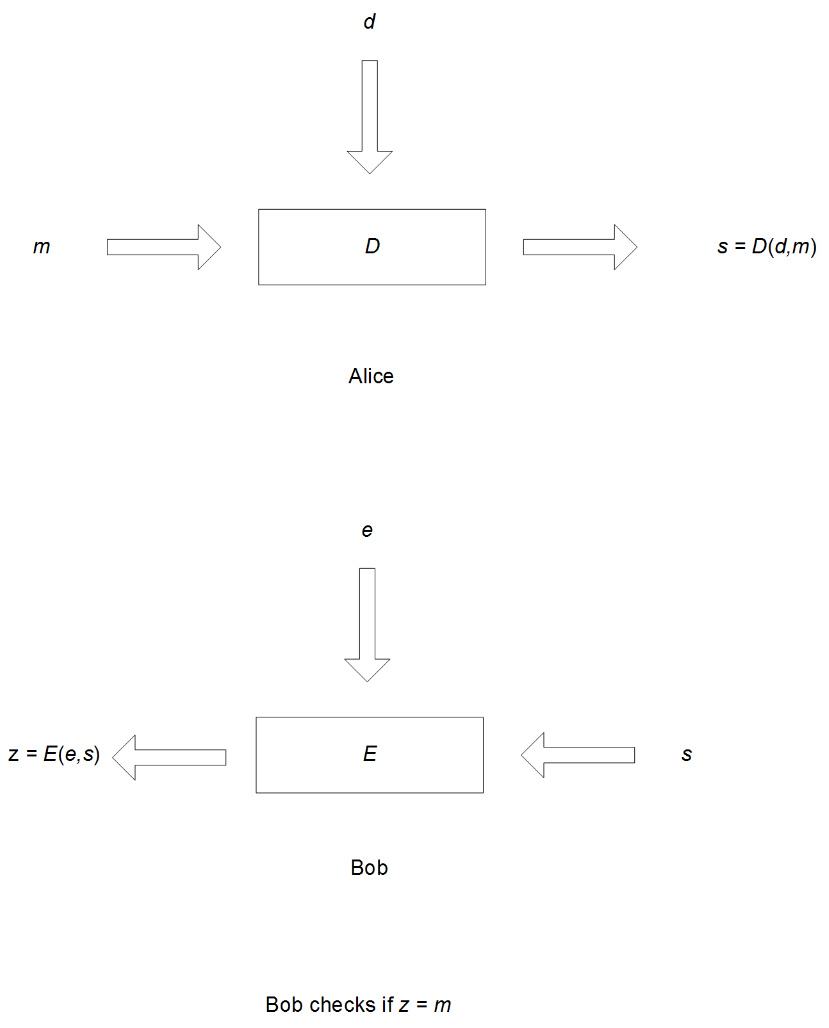


Figure 13: First attempt to produce a digital signature. Note that Bob *does not* run $E$ “in reverse.” It just convenient to have the input be on the right side.

but it is not too difficult. See Fig. 14.) In effect, Mallory started with with a signature and then produced a message “authenticated” with the signature he started with.

What was the source of the difficulty? The sender was permitted to choose what to decrypt, for the purpose of proving own identity (knowing the private decryption key).

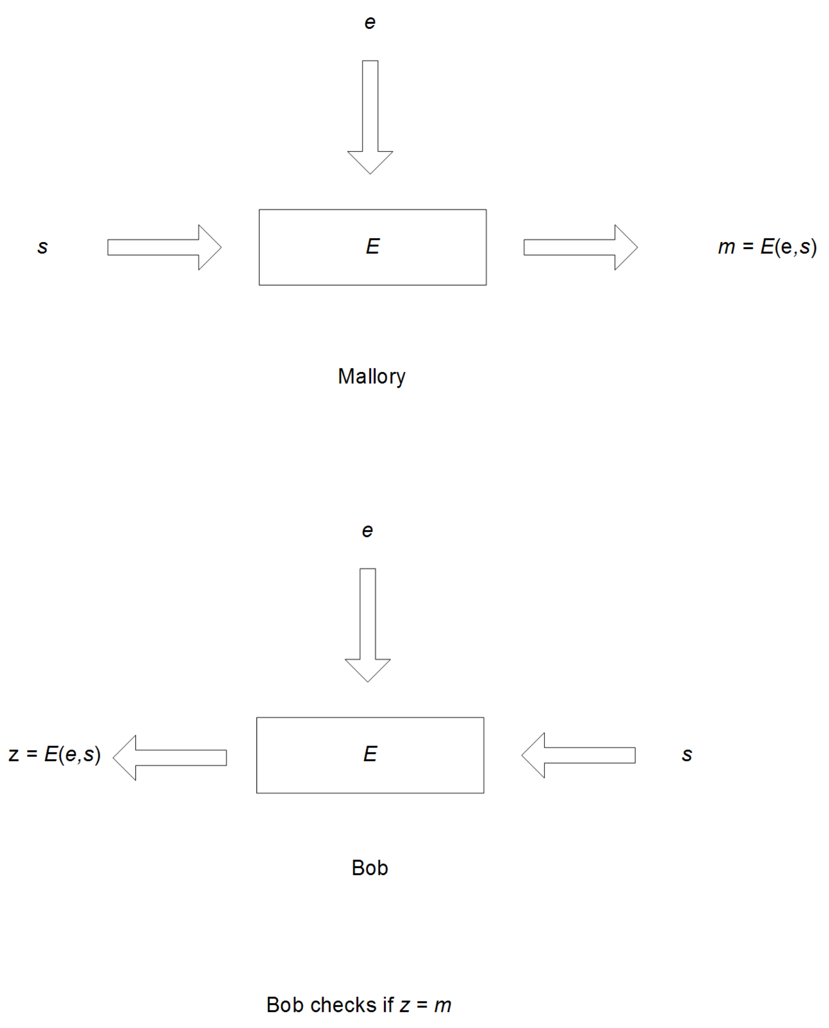


Figure 14: Mallory confuses Bob. Mallory tells Bob that $s$ is the signature and $E(e, s)$ is the (unencrypted) message. Note Bob *does not* run $E$ "in reverse." It just convenient to have the input be on the right side.

Of course, we cannot have the recipient participate in the protocol in practice, as Bob is a passive recipient of email, and additional difficulties would occur, similar to above.

So in some sense, nobody should have complete control over what is to be decrypted to prove Alice's possession of her private key. So we need to produce a protocol in which Alice, in order to produce a signature on a message, will decrypt something that is:

1. Something she cannot "fully" control
2. Is "related" to the message to be signed (otherwise, the same signature could be used for many messages, and more problems crop up).

We will now discuss how it is really done. Look at Fig. 15.

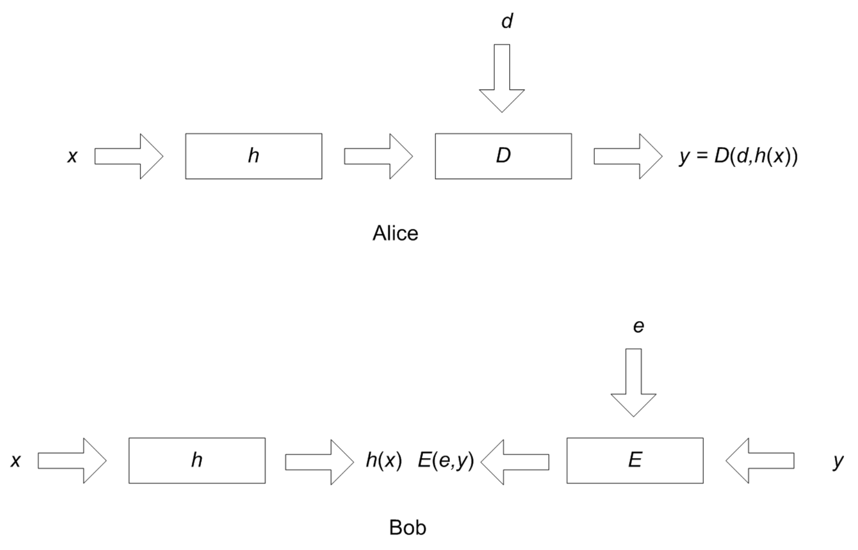


Figure 15: Digital signature production and verification. Note that Bob *does not* run $E$ "in reverse." It is just convenient to have the input be on the right side.

Alice wants to send $x$ and its signature $y$:

1. Alice computes $h(x)$
2. Alice produces $y = D(d, h(x))$ and sends $x$ as messages and $y$ as its signature to Bob
3. Bob computes $h(x)$ and $E(e, y)$. Iff (if and only if) they are equal, he believes that Alice has signed the message.

   Note that Bob cannot just reverse what Alice did as he cannot compute $h^{-1}(E(e, y))$.

This is the standard approach used in digital signatures.

## 4.5 How do I know I am talking to Amazon (take 2)?

We have seen in Sec. 4.4 that it is not a good idea (at least some of the time) for anybody to sign something somebody else sends in. But this is what Alice did in Sec. 4.1. So we would like to modify this and have her sign something else to prove she is Alice, that is decrypt a string that was not given to her but something else. But we also cannot let her choose what to decrypt.

Why? otherwise, Mallory can pretend to be Alice.

So in some sense, what Alice decrypts (to prove her identity) must be something that is not chosen by anybody but comes as a "random" string that has not been seen before, or picked explicitly by anybody. We do something that is functionally "essentially equivalent" to this. Here is the protocol:

1. Bob picks $x$
2. Bob sends $x$ to Alice
3. Alice computes $w = h(x)$
4. Alice "decrypts" $w$ getting $D(d, w) = y$
5. Alice sends $y$ to Bob
6. Bob encrypts $y$ getting $E(e, y) = z$
7. Bob checks if $z = h(x)$. If yes, he believes he is talking to Alice.

What have we accomplished? Neither Alice not Bob picked $w$, which Alice decrypted. We could say that Bob "seeded" the process that created it, but $h$ produced "something random" (this is informal but captures what is actually going on).

## 4.6 Encrypting and signing

We now know how we could both encrypt and sign. So Alice can both encrypt and sign a message to Bob. So Bob is the only one who can read it, be sure it is from Alice, and Alice cannot repudiate the message.

What we describe is correct but not normally used because it is relatively inefficient and requires some tedious "fixing" in practice. For sketch of the protocol, see Fig. 16. In Sec. 4.7, we will describe the common approach, which is a little more complicated, but much more efficient.

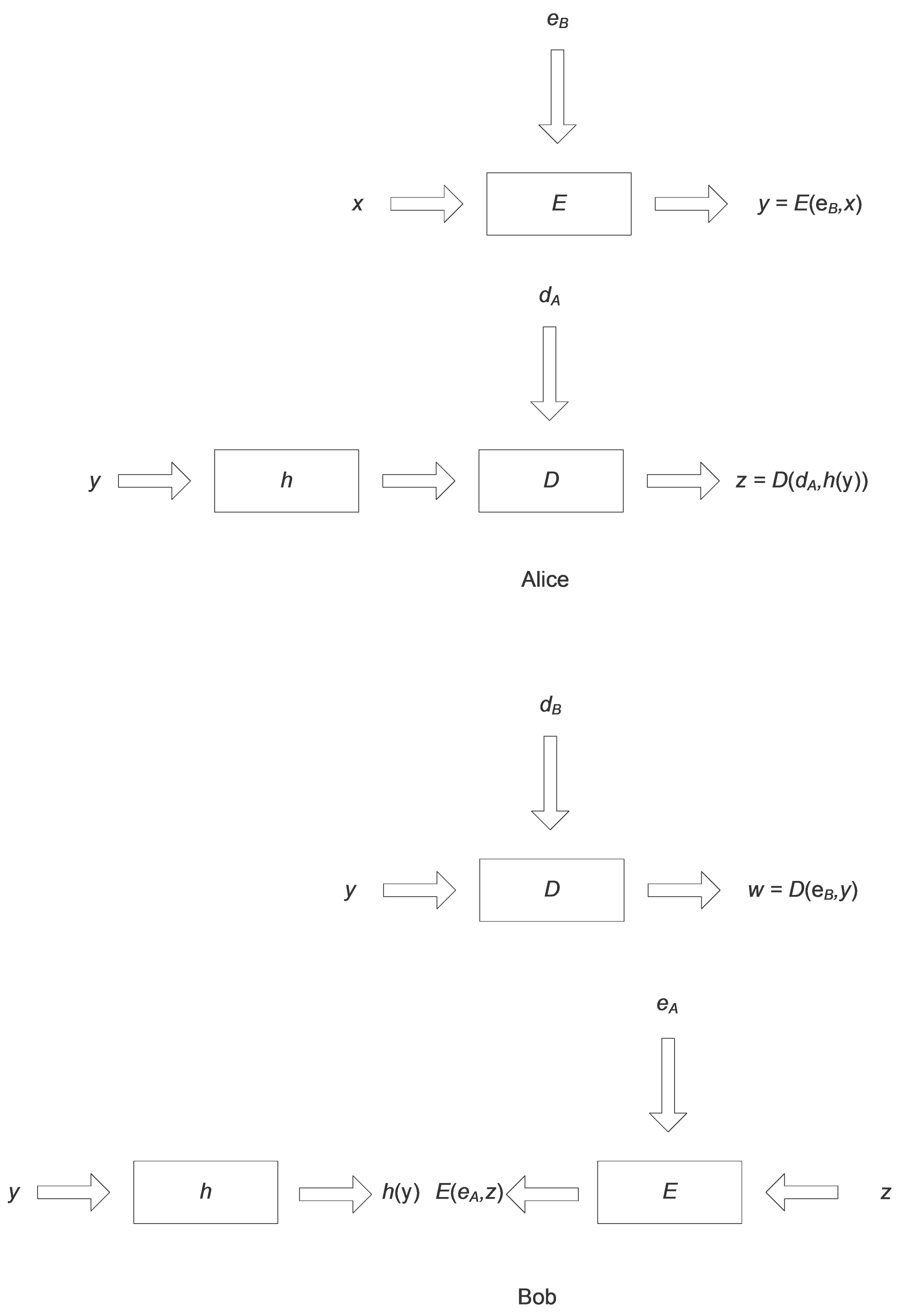


Figure 16: Encrypting and signing using public/private keys and hashing. Alice sends both $y$ (as message) and $z$ as signature.

We assume that both Alice and Bob have (public,private) key pairs, respectively: $(e_{\mathrm{A}}, d_{\mathrm{A}})$ and $(e_{\mathrm{B}}, d_{\mathrm{B}})$.

Alice will encrypt and sign a message to Bob. Bob will decrypt it and verify the signature.

1. Alice chooses a message $x$. She encrypts it with Bob's public key, getting $y = E(e_{\mathrm{B}}, x)$

2. Alice signs $y$ by producing $z = D(d_{\mathrm{A}}, h(y))$

3. Alice sends $y$ and $z$ to Bob as a message and its signature

4. Bob decrypts $y$ with his private key getting $w = D(d_{\mathrm{B}}, y)$

5. He now checks whether $y$ was indeed sent by Alice. He encrypts $z$ with Alice's public key, getting some $u = E(e_{\mathrm{A}}, z)$. This should be $h(y)$, because for any string applying $D$ first and $E$ second with some key pair $(e, d)$ should give the string back. So he computes $h(y)$ and checks whether it is equal to $u$. If it is, he is satisfied that is he knows that $w = x$.

## 4.7 Encrypting and signing (take 2)

Let us now turn to how things are actually done. Public/private key encryption is about 1,000 times slower than symmetric. Therefore, symmetric encryption is preferred for encrypting (long) messages. The problem then appears: how should Alice and Bob agree on a symmetric key.[26] Alice cannot just send it to Bob in the clear! The solution is for Alice to choose a symmetric key for them to communicate and to send this key to Bob using his public key. Look at Fig. 17.

Let us look first at how to encrypt and decrypt messages.

We will use two cryptosystems, public/private (here just to be definite and make notation easier, we will assume it is RSA) and symmetric (here just to be definite and make notation easier, we will assume it is DES). To not be confused what is being used, we will employ "RSA" and "DES" in superscript, so for instance the encrypting machine for DES will be denoted by $E^{\mathrm{DES}}$ and Alice's RSA decryption key by $d_{\mathrm{A}}^{\mathrm{RSA}}$.

The idea is that a symmetric key is used by Alice to encrypt her message to Bob, and Alice picks such a key.

[26] Recall that for symmetric encryption, $d$ is easily computed from $e$ by anybody.

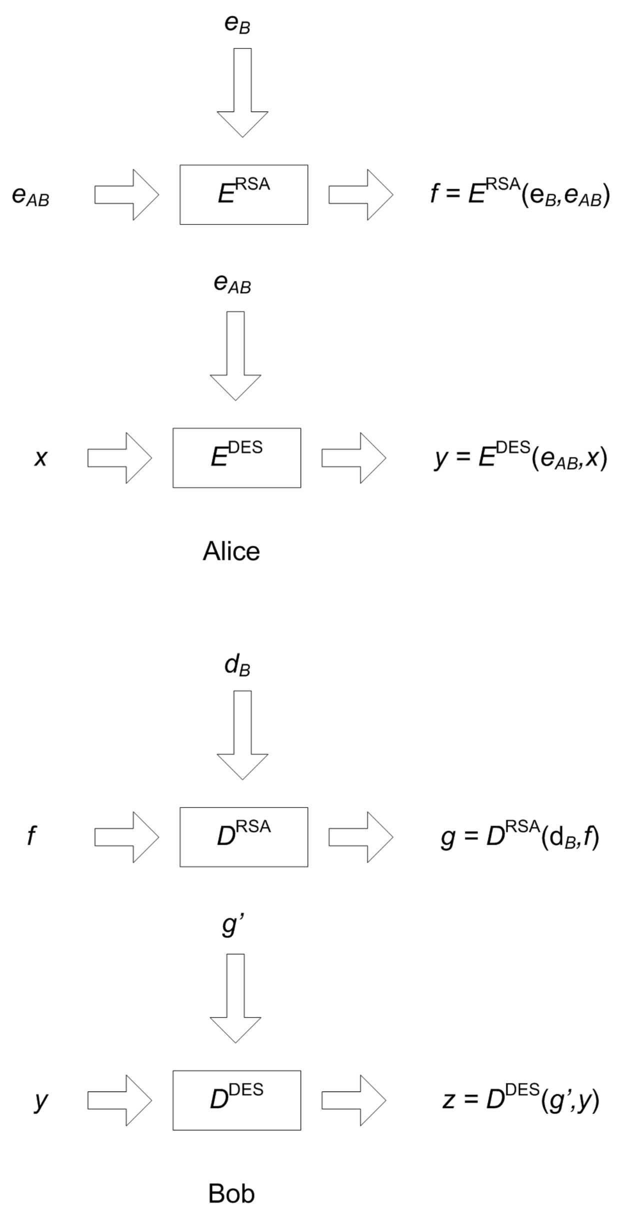


Figure 17: Encrypting using symmetric and public/private keys. In order not to "clutter up" the figure, the superscripts for the keys have been omitted.

The question is, how does she tell Bob what the symmetric key is, if she has never communicated with him before. She encrypts the symmetric key with Bob's public key, and sends this to Bob, in addition to the encrypted message of interest.

Here is the protocol for encryption/decryption

1. Alice picks a secret, symmetric key for encrypting her messages to Bob (only). It is $e_{\mathrm{AB}}^{\mathrm{DES}}$. It is to be used by Alice and Bob to communicate. Nobody else will know it.

2. Alice encrypts this secret key with Bob's public key. Then if she sends it, only Bob will be able to read it. She computes $f = E^{\mathrm{RSA}}(e_{\mathrm{B}}^{\mathrm{RSA}}, e_{\mathrm{AB}}^{\mathrm{DES}})$.

3. She uses $e_{\mathrm{AB}}^{\mathrm{DES}}$ to encrypt a message $x$ intended for Bob. She computes $y = E^{\mathrm{DES}}(e_{\mathrm{AB}}^{\mathrm{DES}}, x)$

4. She sends $f$ and $y$ to Bob.

5. Bob wants to find out what is the DES decryption key he is supposed to use. He computes $g = D^{\mathrm{RSA}}(d_{\mathrm{B}}^{\mathrm{RSA}}, f)$. $g$ is in fact the same as $e_{\mathrm{AB}}^{\mathrm{DES}}$. This is the encryption key, he needs the decryption key. The decryption key is easily computable (recall Caesar's cipher), usually in fact the same as encryption key. Let us call this easily derivable decryption key $g'$.

6. Bob uses the decryption key to decrypt the message, getting $z = D^{\mathrm{DES}}(g', y)$. And $z$ is the original $x$.

Signing is done as in Sec. 4.6. The hash of $y$ is short, so can be "decrypted" using the inefficient $D^{\mathrm{RSA}}$ with $d_{\mathrm{A}}^{\mathrm{RSA}}$.

Using what we already know, we can extend this protocol so that Alice can also sign her message.

## 4.8 Digital certificates

See also Forouzan pp. 454–458. and `http://en.wikipedia.org/wiki/X.509`.

There is one additional thing that we should pay attention to now. How does Alice know that she has the correct value of $e_{\mathrm{B}}^{\mathrm{RSA}}$? If she knows Bob, perhaps he told her, but what if she gets it from the Web? How does she know it is correct? She could ask him for a fingerprint, but this is tedious and requires "out of channel" interactions, such as phone calls to verifiable phone numbers, etc. There are various methods, but we will focus now on the most formal one: digital certificates.[27]

[27] I do not have a digital certificate, so I have to use fingerprints.

Recall that Trent is a trusted authority.[28] So if he assures us that Bob's public key is $e_{\mathrm{B}}^{\mathrm{RSA}}$, then of course we will believe him. And how can he assure us: he signs a statement (a message) that Bob's public key is $e_{\mathrm{B}}^{\mathrm{RSA}}$.[29]

We assume that everybody knows Trent's public key, so his signature can be verified, which is in fact true.[30]

Here is a protocol for creating a digital certificate:

1. Bob generates a public key for himself: $e_{\mathrm{B}}^{\mathrm{RSA}}$.[31]
2. Bob goes to Trent and identifies himself reliably, by perhaps showing him a passport, and also presents to him his $e_{\mathrm{B}}^{\mathrm{RSA}}$.
3. Trent creates a string $x = \mathrm{Bob}\|e_{\mathrm{B}}^{\mathrm{RSA}}$, that is concatenation of Bob's name with his public key
4. Trent signs $x$ with his private key, getting $y$.[32] We know how to do it—Trent decrypts $h(x))$ with $d_{\mathrm{T}}^{\mathrm{RSA}}$. The two strings together form a digital certificate that Bob has: $(x, y)$, or perhaps their concatenation: $x\|y$.

The resulting digital certificate is depicted in Fig. 18

| | | |
|---|---|---|
| Bob | $e_{\mathrm{B}}$ | $D(d_{\mathrm{T}},h(\mathrm{Bob}\|e_{\mathrm{B}}))$ |

Figure 18: Bob's digital certificate

## 4.9 Putting it together

We now know enough to understand the procedure for how Alice sends secure email to Bob. It is just reiterating what we already know. Later in the course we will go into more details, for instance we will talk about something similar, PGP, see Forouzan, Section 16.2, pp. 470–492, though he has too

[28] In reality, it is EMC www.rsa.com, Verisign http://www.verisign.com/, Thawte http://www.thawte.com/, or similar.

[29] Actually decrypts the hash of such as statement.

[30] More precisely, "all" computers know it.

[31] We will see how this is done later. In fact, what he does is to generate the pair $(e_{\mathrm{B}}^{\mathrm{RSA}}, d_{\mathrm{B}}^{\mathrm{RSA}})$.

[32] In a real certificate more information is stored as part of $x$, for example expiration date—all of this is not a core issue.

many details for us. See also, GnuPG http://www.gnupg.org/. I will also assign downloading and experimenting with this software.

Alice and Bob have obtained digital certificates from Trent, we know how it is done, see Sec. 4.8. Perhaps they got it for free, using http://www.thawte.com/secure-email/personal-email-certificates/index.html.

Alice wants to send a message securely to Bob.

1. She obtains and verifies Bob's public key, perhaps:
   (a) She gets Bob digital certificate.
   (b) She confirms that it indeed is his digital certificate. How? She "disassembles" it (just cuts it into pieces) and checks that Trent indeed has signed the association of Bob with a public key.
2. She chooses a random symmetric key and
   (a) Encrypts it with the Bob's public key. See Sec. 4.7.
   (b) Uses the symmetric key to encrypt the message to Bob. Again, see Sec. 4.7.
3. Alice decrypts the hash of her encrypted message with her private key, see Sec. 4.4 and Sec. 4.7.
4. Alice sends what she has produced to Bob.
5. Bob decrypts her message and verifies her signature.

## 4.10 Classifying security

Security can be roughly classified as follows:

1. *Absolute (perfect):* Impossible to break given unlimited resources
2. *Not Absolute*:
   (a) *Conditional:* Given reasonable resources can break only with very small probability if some mathematical problem is difficult to compute/solve (but in general we only have beliefs that specific mathematical problems are difficult to compute/solve)— we will see some of these problems later in the course.
   (b) *Heuristic:* Given reasonable resources can break only with very small probability (functions used are very strange and combinatorially confusing—we will see some of them later in the course).

## 4.11 Key ideas

1. One way function and one way-hash; using the latter for message integrity and various applications
2. Protocol for signing
3. Using a public/private system for transmitting a secret key (for using a symmetric system for encryption)
4. Digital certificates
5. Classification of security

## 4.12 Please read

All the material in Forouzan referred to in this section other than Section 16.2 (PGP).

# 5 Probability, one-time pad security, and entropy

## 5.1 Probability

We will have a simple overview of basic probability on finite sets. Discussion with be informal. (Some of what we do may not be correct for infinite sets.)

As good examples, we will have several objects:

1. A fair coin $C_0$, which with probability 0.5 each gives us H (heads) or T (tails)

2. An unfair coin $C_1$, which with probability 0.99 gives us H and with probability 0.01 gives us T

3. An extremely unfair coin $C_2$, which with probability 1 gives us H and with probability 0 gives us T [33]

4. A fair die $D_0$, which with probability 1/6 gives us each of $1, \ldots, 6$.

**Caution:** An event of probability 0 can still happen—it is just that "the probability of this is smaller than any positive number," but if we assume informally that it never happens we will be OK, because we are dealing with finite sets.[34]

### 5.1.1 Basic concepts

We will assume that we have a (finite) set $X$ and for each $x \in X$, we have a number $p_x$, such that $0 \leq p_x \leq 1$ and $\sum_{x \in X} p_x = 1$. $p_x$ is the probability that if we pick "randomly" from $X$, we will get $x$. $\mathcal{X}$ is $X$ together with $\{p_x \mid x \in X\}$. This is called a *random variable.* We will also write $\Pr[x]$ or $\Pr[\mathcal{X} = x]$ for $p_x$.

Sometimes a real-valued function, say $f$ is defined on $X$. Then the *expectation* of $f$ is $\text{Ex}[f] = \sum_{x \in X} \Pr[x] \cdot f(x)$.[35]

[33] How this is done is not important for us.

[34] But once you go to infinite sets you need to be much more careful. For example, if you pick a real number uniformly from the interval $[0, 1]$ then of course each time you do it, you will get some real number. But the probability of picking any *specific* number must be 0.

[35] Expectation essentially means "average." It does not mean that this is value that is expected to happen. For example the expectation of the number of children per woman in the US is about 2.2.

For example, if we toss $C_1$ and $f(\mathrm{H}) = 2$ and $f(\mathrm{T}) = 3$, then the expectation (how much money we make on the average per toss if we get paid $f$) is $0.99 \cdot 2 + 0.01 \cdot 3$.

If we want to use pictures, then we could draw a square of area 1, and then for each $x$, $p_x$ will be the area of an appropriate "part" of the square. See Fig. 19, Fig. 20, Fig. 21, Fig. 22.

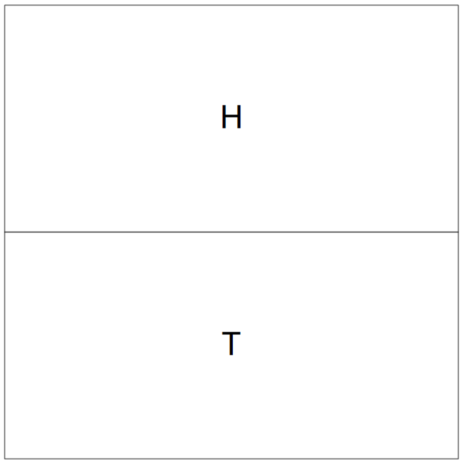


Figure 19: $C_0$.

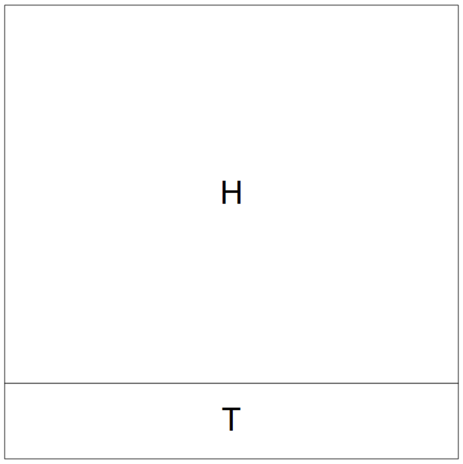


Figure 20: $C_1$. Imagine that the lower stripe is only 1/100th of the total height.

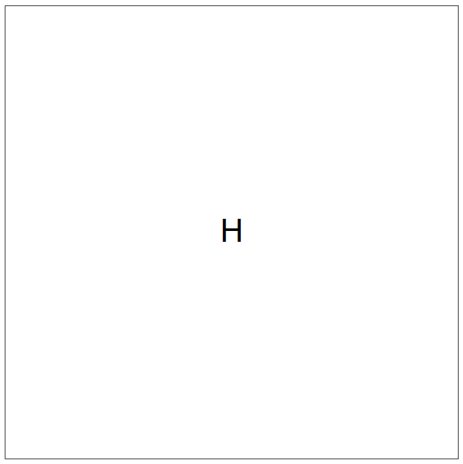


Figure 21: $C_2$.

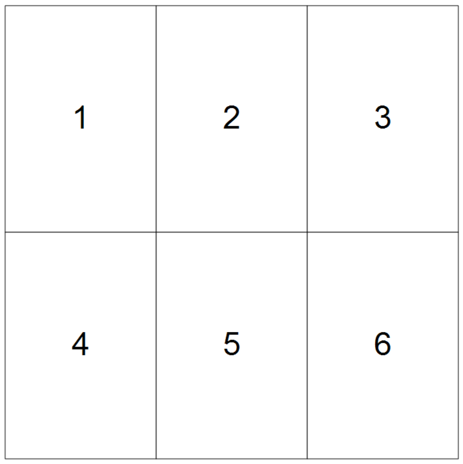


Figure 22: $D_0$.

$E \subseteq X$ is called an *event*. So, for instance {H} is an event, but in this case we may as well talk about H.

An example of a more interesting event for $D_0$ is $E = \{2, 5\}$.

In general, we have:

$$\Pr[x \in E] = \sum_{x \in E} \Pr[x]$$

So for our event: $\Pr[x \in \{2, 5\}] = 1/3$. Compare with the areas in the corresponding figure.

If we have two events $A$ and $B$, then of course we can talk about events, $\bar{A}$, $A \cup B$ and $A \cap B$.[36] For an example of complementary events see Fig. 23, for an example of disjoint events, see Fig. 24; and for an example of not disjoint events, see Fig. 25.

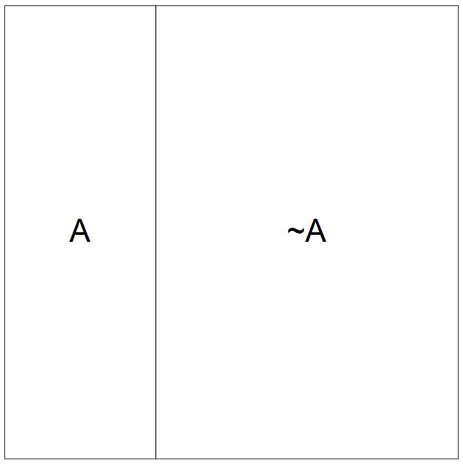


Figure 23: complementary events.

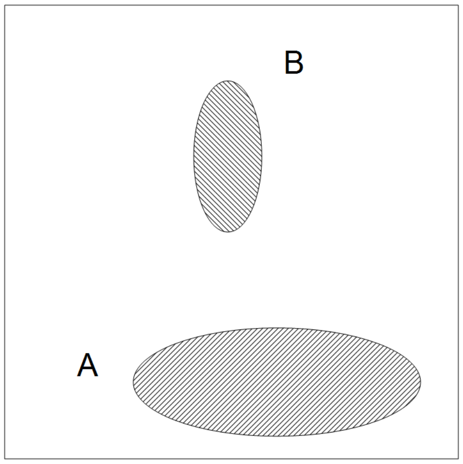


Figure 24: Disjoint events.

[36] $\bar{A} = X \setminus A$ (also written as $X - A$). Sometimes $\sim A$ is used to denote the complement of $A$.

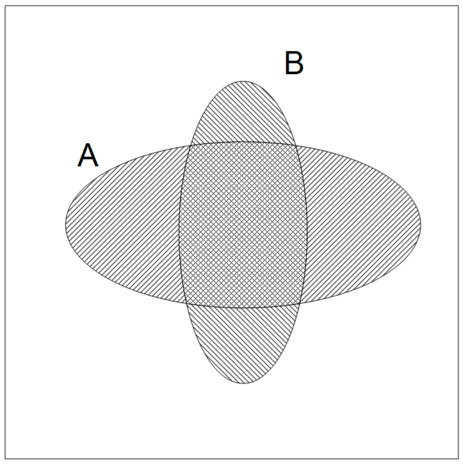


Figure 25: Not disjoint events.

Let us now toss first $D_0$ and then $C_0$. Look at Fig. 26. Look at Fig. 27. It shows the probabilities explicitly as areas.

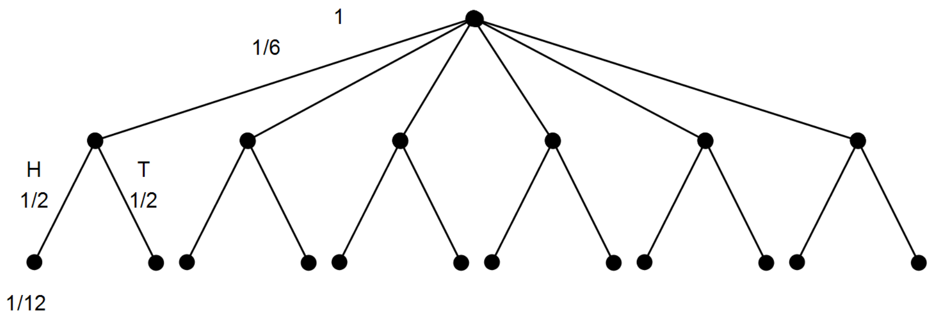
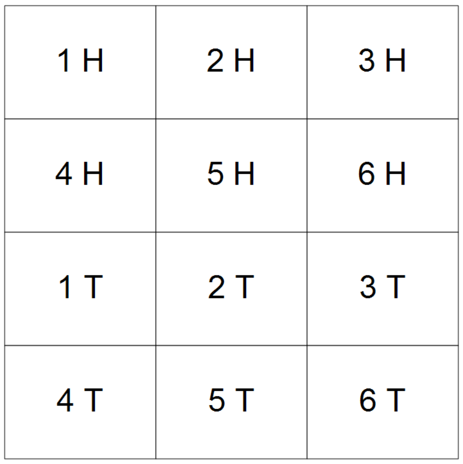


Figure 26: Tossing $D_0$ and then $C_0$. Only some of the probabilities are written out. The probability is written below the result of the random variable.

Let us consider what is the probability of getting the pair $(1, \mathrm{H})$ (this probability is frequently denoted as $\Pr[1, \mathrm{H}]$, meaning we get both 1 and H. Of course, $\Pr[1, \mathrm{H}] = \Pr[\mathrm{H}, 1]$

The result of tossing a die says nothing about the result of tossing the coin. Informally speaking for now, these two variables/events are *independent*. The probability of getting this result is just the product of probabilities of getting each of them separately, in this case $1/6 \cdot 1/2 = 1/12$.

Similarly, if we first toss $D_0$ and then $C_1$, though the actual numbers are,



| | | |
|---|---|---|
| 1 H | 2 H | 3 H |
| 4 H | 5 H | 6 H |
| 1 T | 2 T | 3 T |
| 4 T | 5 T | 6 T |

Figure 27: Probabilities of tossing $D_0$ and $C_0$.

of course, different.

We had a sequence of random variables. Let us look at a slightly more interesting one. We toss $C_0$, If we get H we toss $C_0$; if we get T, we toss $C_1$. What are the probabilities of getting HH, HT, etc. We just multiply probabilities, see Fig. 28. We get $\Pr[\mathrm{T},\mathrm{H}] = 0.5 \cdot 0.99$, etc.

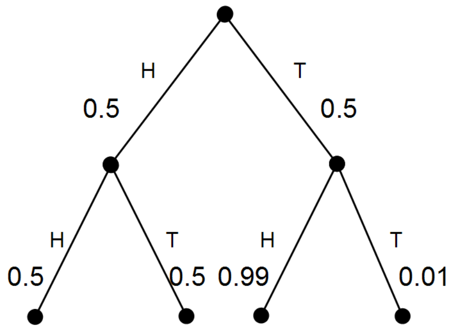


Figure 28: Tossing $C_0$ first and then $C_0$ or $C_1$.

### 5.1.2 Conditional probability and Bayes' theorem

Let us now talk about the very important concepts of independent variables and conditional probabilities.

Everything we need to know can be understood by carefully examining Fig. 25. For examples, it is good to look at the more specific figures dealing with the die and the various coins.

Assume that as the result of getting a random variable (value) (e.g., tossing a coin, or die, or both) we are in event $B$. What is the probability that we are also in event $A$? Looking at the figure (and of course assuming that $\Pr(B) \neq 0$), this is

$$(1) \quad \Pr[A \mid B] = \frac{\Pr[A, B]}{\Pr[B]}$$

Note that this is really

$$\frac{\text{area}(A \cap B)}{\text{area}(B)}$$

$\Pr[A \mid B]$ denotes conditional probability: what is the probability of $A$ given $B$. Recall that $\Pr[A, B]$ denotes the probability of both $A$ and $B$, that is we find ourselves in $A \cap B$.

Now, let's look at some examples:

1. We toss $D_0$. What is $\Pr[x > 1]$? Looking at Fig. 22, we see this is 5/6 (5 rectangles out of 6).

2. We toss $D_0$. What is $\Pr[x \text{ is even}]$? Looking at Fig. 22, we see this is 3/6

3. We toss $D_0$. What is $\Pr[x > 1 \text{ and is even}]$? Looking at Fig. 22, we see this is 3/6

4. We toss $D_0$. What is $\Pr[x \text{ is even} \mid x > 1]$? Looking at Fig. 22, we see this is $(3/6)/(5/6) = 3/5$

5. We toss $D_0$. What is $\Pr[x > 1 \mid x \text{ is even}]$? Looking at Fig. 22, we see this is $(3/6)/(3/6) = 1$

Let us now return to the case of first tossing $D_0$ and then tossing $C_0$, see Fig. 26 and Fig. 27.

The results of the toss seem independent (and they indeed are). Knowing the result of tossing the die does not tell us anything about the result of tossing the coin. Let us compute $\Pr[\text{H} \mid 1]$. It is $(1/12)/(2/12) = 1/2$.

In this case, $\Pr[\text{H} \mid 1] = \Pr[\text{H}]$. This is essentially (but not quite) the definition of the two events getting H and getting 1, being independent: the result of tossing the coin and the result of tossing the die are independent of each other.

Let us now prove a very important result:

**Theorem 1.** (Bayes) http://en.wikipedia.org/wiki/Bayes'_theorem (this is much more than we need to know, though the "bowls and cookies" example is nice).

$$\Pr[A \mid B] = \frac{\Pr[B \mid A] \cdot \Pr[A]}{\Pr[B]} \tag{2}$$

Of course, we assume that $\Pr[B] \neq 0$, as we divide by it. □ [37]

*Proof.* We know that

$$\Pr[A \mid B] = \frac{\Pr[A, B]}{\Pr[B]}$$

and therefore

$$\Pr[A \mid B] \cdot \Pr[B] = \Pr[A, B] \tag{3}$$

Note that equation holds not only when $\Pr[B] \neq 0$, but also when $\Pr[B] = 0$ since then both sides of the equation are 0.

Also, by exchanging $A$ and $B$, we get

$$\Pr[B \mid A] \cdot \Pr[A] = \Pr[B, A] \tag{4}$$

But of course,

$$\Pr[B, A] = \Pr[A, B]$$

and therefore from Equations 3 and 4

$$\Pr[A \mid B] \cdot \Pr[B] = \Pr[B \mid A] \cdot \Pr[A]$$

Therefore:

$$\Pr[A \mid B] = \frac{\Pr[B \mid A] \cdot \Pr[A]}{\Pr[B]}$$

□

[37] We use the symbol □ as a general "end" marker, and not just as a shorthand for "Q.E.D." http://en.wikipedia.org/wiki/Q.E.D.

**Definition 2.** (Formally,) $A$ and $B$ are *independent* iff

$$\text{(5)} \quad \Pr[A, B] = \Pr[A] \cdot \Pr[B]$$

□

Let us go immediately to the intuition.

**Theorem 2.** If $\Pr[B] \neq 0$, then $A$ and $B$ are independent iff

$$\text{(6)} \quad \Pr[A \mid B] = \Pr[A]$$

(This could have served as a more intuitive definition. What it tells us that knowing that we are "in" $B$ does not help us in "predicting" whether we are also in $A$. The reason we had definition 5 was that this could be written even for the case when $\Pr[B] = 0$.)

□

*Proof.* 1. Assume Eq. 5 [38]

$$[\Pr[A, B] = \Pr[A] \cdot \Pr[B]]$$

We know that in general, Eq. 1

$$\left[\Pr[A \mid B] = \frac{\Pr[A, B]}{\Pr[B]}\right]$$

Substituting what is assumed we get:

$$\Pr[A \mid B] = \frac{\Pr[A] \cdot \Pr[B]}{\Pr[B]} = \Pr[A]$$

and we get Eq. 6.

2. Assume Eq. 6.

$$[\Pr[A \mid B] = \Pr[A]]$$

[38] I put in bracket an equation that we numbered before, but I find convenient repeating, just like the next equation.

Substituting this into Eq. 2

$$\left[\Pr[A \mid B] = \frac{\Pr[B \mid A] \cdot \Pr[A]}{\Pr[B]}\right]$$

we get

$$\Pr[A] = \frac{\Pr[B \mid A] \cdot \Pr[A]}{\Pr[B]}$$

Using Eq. 4

$$[\Pr[B \mid A] \cdot \Pr[A] = \Pr[B, A]]$$

we get

$$\Pr[A] = \frac{\Pr[B, A]}{\Pr[B]}$$

But as $\Pr[B, A] = \Pr[A, B]$

we get

$$\Pr[A] \cdot \Pr[B] = \Pr[A, B]$$

and we get Eq. 5

□

**Observation 1.**

(7) $\Pr[B] = \Pr[B \mid A] \cdot \Pr[A] + \Pr[B \mid \bar{A}] \cdot \Pr[\bar{A}]$.

Look at Fig. 29. We will discuss only the case where $B$ overlaps both $A$ and $\bar{A}$, as in the figure; the other cases are even simpler.

Using Eq. 1 and its complement,

$$\Pr[A \mid B] = \frac{\Pr[A, B]}{\Pr[B]} \qquad \text{and} \qquad \Pr[\bar{A} \mid B] = \frac{\Pr[\bar{A}, B]}{\Pr[B]}$$

the claim is equivalent to

$$\Pr[B] = \Pr[B, A] + \Pr[B, \bar{A}]$$

which of course is true.

□

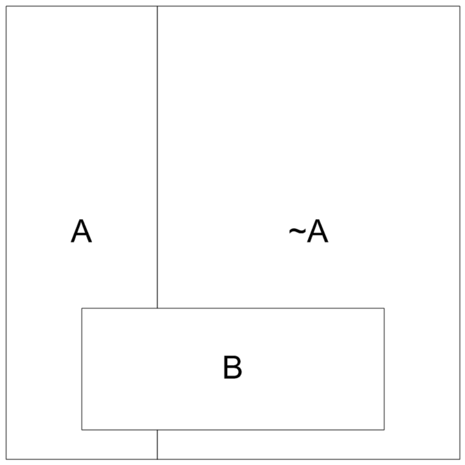


Figure 29: $B$ and $A$ and conditional probabilities.

**Corollary 1.** From

From Eq. 2 and Eq. 7, we immediately get:

$$\Pr[A \mid B] = \frac{\Pr[B \mid A] \cdot \Pr[A]}{\Pr[B \mid A] \cdot \Pr[A] + \Pr[B \mid \bar{A}] \cdot \Pr[\bar{A}]} \tag{8}$$

□

If you want, you can look at `http://www.mathgoodies.com/lessons/vol6/challenge_vol6.html`.

## 5.2 Practicing Bayesian thinking

We now have the machinery to analyze the one-time pad cryptosystem. But before doing this, we will look at two interesting examples to practice what we have learned about "Bayesian thinking."

**Example 1.** On the table, there are two coins, one $C_0$ and the other $C_2$. Without looking, you pick one and toss it twice. You get the sequence HH. What is the probability that you picked $C_2$. Look at Fig. 30.

Let $C$ be the result of choosing the coin, so either $C_0$ or $C_2$. So, using Eq. 2, which is

$$\left[\Pr[A \mid B] = \frac{\Pr[B \mid A] \cdot \Pr[A]}{\Pr[B]}\right]$$

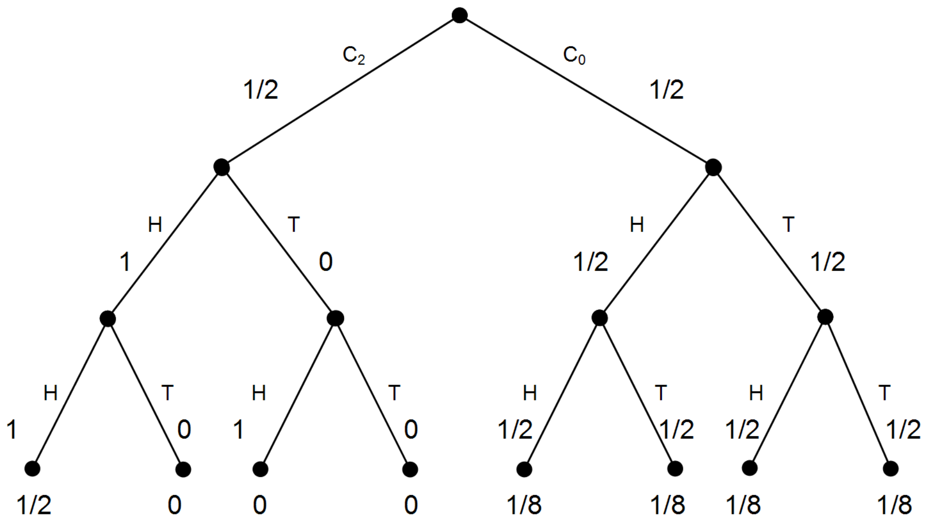


Figure 30: Tossing one of two coins

we get

$$\Pr[C = C_2 \mid \text{HH}] = \frac{\Pr[C = C_2] \cdot \Pr[\text{HH} \mid C_2]}{\Pr[\text{HH}]}$$

1. $\Pr[C = C_2] = 1/2$
2. $\Pr[\text{HH} \mid C_2] = 1$
3. $\Pr[\text{HH}] = 1/8 + 1/2$, from the figure

So, the final result is $4/5$.

**Example 2.** (Optional: this is more than we need for the class, but is actually interesting and potentially useful in many settings.) Various businesses advertise tests such as total CT scan for symptomless people. After all, something could be lurking in the body that is dangerous but unknown to the person. But there could be *false positive* when the test declares a healthy person to be sick.[39] Let us consider a scenario.

There exists a horrible disease and an imperfect test for it. There also exists an imperfect drug for the disease. Let us look at some numbers.

[39]Or potentially sick, which may require additional, possibly life-threatening tests, like biopsies.

1. The probability of having the disease is 0.001 (very small)
2. If the test is administrated to a sick person, it will correctly determine with probability 0.98 that the person is sick, and will declare with probability 0.02 that the person is healthy (very good test)
3. If the test is administrated to a healthy person, it will correctly determine with probability 0.99 that the person is healthy, and will declare with probability 0.01 that the person is sick (very good test)
4. If the drug is administered to a sick person, the person will be cured instantenously, if it is not, the person will die (very good drug)
5. If the drug is administered to a healthy person, nothing will happen with probability 0.7, but the person will die instantenously with probability 0.3 (don't administer the drug to healthy people)

Should I take the test (and act on it, otherwise, why take it)?
Let us define some events:

1. $A$: I am healthy
2. $B$: the test came positive

What I am interested in are false positives, because if the test came positive, and I am healthy, and I take the drug, I have some probability of dying. What I want to know is what is the probability that I am actually healthy even though the test comes positive, so I want to know: $\Pr[A \mid B]$. Very easy, let's substitute into Eq. 8, which is

$$\left[\Pr[A \mid B] = \frac{\Pr[B \mid A] \cdot \Pr[A]}{\Pr[B \mid A] \cdot \Pr[A] + \Pr[B \mid \bar{A}] \cdot \Pr[\bar{A}]}\right]$$

We need to compute the terms appearing there:

1. $\Pr[B \mid A] = 0.01$
2. $\Pr[A] = 0.999$
3. $\Pr[B \mid \bar{A}] = 0.98$
4. $\Pr[\bar{A}] = 0.001$

So, plugging it in:

$$\Pr[\text{healthy} \mid \text{test positive}] = \frac{0.01 \cdot 0.999}{0.01 \cdot 0.999 + 0.98 \cdot 0.001} = 0.91$$

We have many false positives. Let us now see what happens when every person who gets a positive test result is treated with the drug. It is best to look at Fig. 31

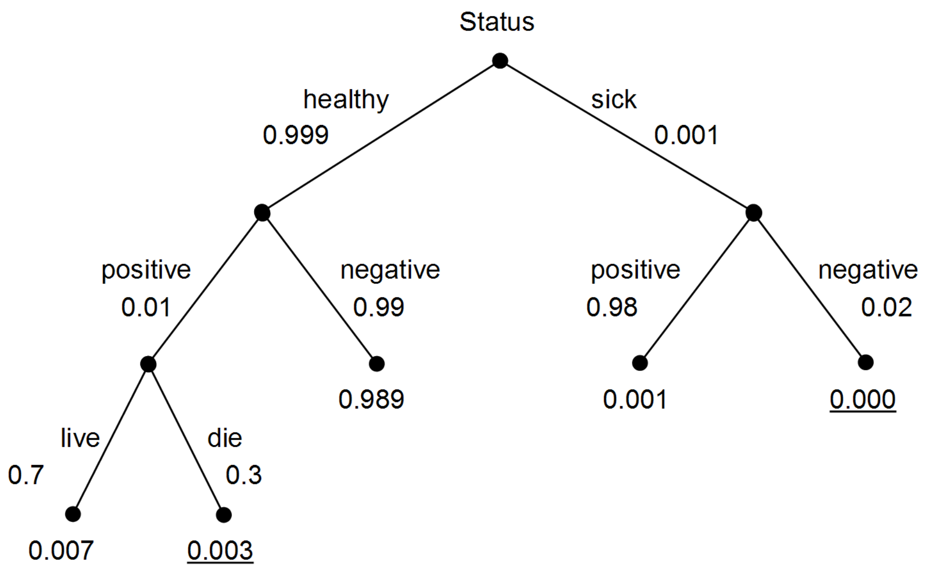


Figure 31: The result of testing and treating. The probability at a leaf is the product of probabilities along the path from the root, computed up to three decimal places. The cases resulting in death are underlined.

We see that if people are tested and treated, three times more will die than in the case when nobody is tested, as we have the probability of death of $0.003 + 0.000$, as opposed to 0.001.

(For another version of this example, see http://en.wikipedia.org/wiki/Bayes'_theorem#Example_.232:__Drug_testing.)

□

## 5.3 Analyzing the one-time pad cryptosystem

Let us now analyze the one-time-pad cryptosystem.

Alice sends 1-bit messages to Bob using one-time pads. Eve listens to the ciphers in transit.

Everybody knows that any message (not cipher) sent by Alice

$$m = \begin{cases} 0 & \text{with probability } 1/3 \\ 1 & \text{with probability } 2/3 \end{cases}$$

We are not saying that she picks messages randomly with the specified probability (although maybe she does) it is just that she happens to say 1 more than she happens to say 0 (she is nice and likes saying "yes" more than saying "no"). So her message stream could have been something biased towards 1, such as: 1 1 0 1 1 0 0 0 1 1 1 1 0 1 1 . . . .

We will examine what can Eve learn based on how Alice choses the one-time pad.

### 5.3.1 Using a biased (unfair) coin to get $k$

Assume that Alice has a rather unfair coin she uses to get the bits for the one-time pad, and she chooses:

$$k = \begin{cases} 0 & \text{with probability } 9/10 \\ 1 & \text{with probability } 1/10 \end{cases}$$

The question is what can Eve deduce about $m_i$ given $c_i$ (which she sees), or what is the probabilistic distribution of $m_i$ given $c_i$. Or mathematically, she wants to know $\Pr[m_i = 0 \mid c_i = 0]$ (and other combinations, such as $\Pr[m_i = 0 \mid c_i = 1]$). As we are talking about a single message of one bit (other bits treated the same), let's drop the subscript.

Let us look at Fig. 32. By examining it, we will figure out some probabilities.

If Eve sees $c = 0$, which happens $9/30 + 2/30 = 11/30$ of the time, the probability that $m = 0$ was

$$\Pr[m = 0 \mid c = 0] = \frac{\Pr[m = 0, c = 0]}{\Pr[c = 0]} = \frac{9/30}{11/30} = \frac{9}{11}$$

which is much better estimate than

$$\frac{1}{3}$$

which she has without knowing that $c = 0$.

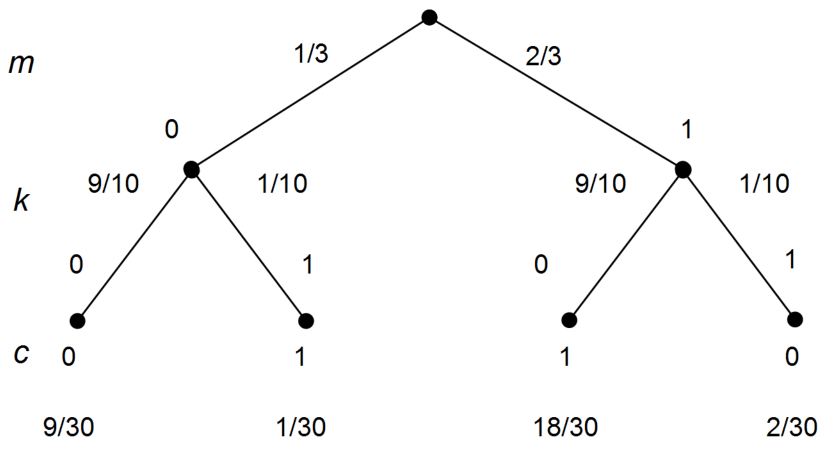


Figure 32: One-time pad. Alice uses a biased coin to select the key $k$.

This is not really surprising, as in $9/10$ of the cases $c = m$. Of course, we would get the same type of result if

$$k = \begin{cases} 0 & \text{with probability } 1/10 \\ 1 & \text{with probability } 9/10 \end{cases}$$

because in $9/10$ of the cases $m = c \oplus 1$.

We could have gotten the same result using Eq. 2.

$$\Pr[m = 0 \mid c = 0] = \frac{\Pr[m = 0] \cdot \Pr[c = 0 \mid m = 0]}{\Pr[c = 0]}$$

$\Pr[m = 0] = 1/3$, given to us

$\Pr[c = 0 \mid m = 0] = \Pr[k = 0 \mid m = 0]$, because given $m = 0$, it follows that $c = 0$ iff $k = 0$. But also $\Pr[k = 0 \mid m = 0] = \Pr[k = 0]$, because $k$ is chosen independently of $m$.

Therefore $\Pr[c = 0 \mid m = 0] = \Pr[k = 0] = 9/10$

$c = 0$ happens when $(m = 0) \wedge (k = 0)$ or $(m = 1) \wedge (k = 1)$. But these are disjoint (you cannot have simultaneously $k = 0$ and $k = 1$), and therefore

$\Pr[c = 0] = \Pr[m = 0, k = 0] + \Pr[m = 1, k = 1]$ (we add probabilities because the events are disjoint, see Fig. 24).

But because the choices of $m$ and of $k$ are independent, $\Pr[c = 0] = \Pr[m = 0] \cdot \Pr[k = 0] + \Pr[m = 1] \cdot \Pr[k = 1]$. (Product, because independent.)

Putting it all together,

$\Pr[c = 0] = (1/3)(9/10) + (2/3)(1/10) = 11/30$

So,

$\Pr[m = 0 \mid c = 0] = (1/3)(9/10)/(11/30) = 9/11.$

### 5.3.2 Using an unbiased (fair) coin to get $k$

Assume now that Alice has a fair coin she uses to get the bits for the one-time pad, and she chooses:

$$k = \begin{cases} 0 & \text{with probability } 1/2 \\ 1 & \text{with probability } 1/2 \end{cases}$$

The question is what can Eve deduce about $m_i$ given $c_i$ (which she sees), or what is the probabilistic distribution of $m_i$ given $c_i$. Or mathematically, she wants to know $\Pr[m_i = 0 \mid c_i = 0]$ (and other combinations, such as $\Pr[m_i = 0 \mid c_i = 1]$). As we are talking about a single message of one bit (other bits treated the same), let's drop the subscript.

Let us look at Fig. 33.

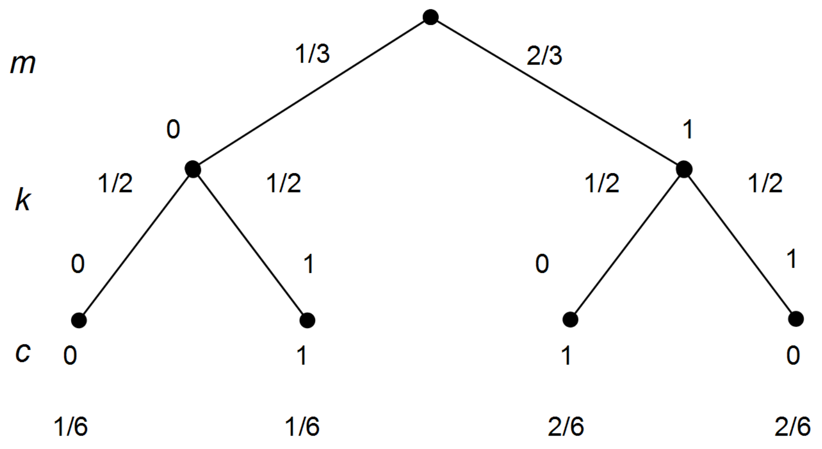


Figure 33: One-time pad. Alice uses an unbiased (fair) coin to select the key $k$.

If Eve sees $c = 0$, which happens $1/6 + 2/6 = 1/2$ of the time, the probability that $m = 0$ was

$$\Pr[m = 0 \mid c = 0] = \frac{\Pr[m = 0, c = 0]}{\Pr[c = 0]} = \frac{1/6}{1/2} = \frac{1}{3}$$

which is not a better estimate than

$$\frac{1}{3}$$

which she has without knowing that $c = 0$.

If Eve sees $c = 1$, which happens $1/6 + 2/6 = 1/2$ of the time, the probability that $m = 0$ was again

$$\frac{1/6}{1/2} = \frac{1}{3}$$

So knowing the cipher does not help. We could have proven this also using Eq. 2, as before.

### 5.3.3 General statement

**Definition 3.** A cryptosystem is *perfect* when knowing $c$ does not help in figuring out anything additional that was not known before about $m$ (given unbounded computational resources), or more formally,

$$\Pr[m = m_0 \mid c = c_0] = \Pr[m = m_0] \tag{9}$$

where,

1. $c_0 = E(e, m_0)$
2. $E$ known
3. $e$ unknown (in one-time-pad, we used the notation $k$ for $e$)

□

**Theorem 3.** The one-time pad cryptosystem (with a perfect, unbiased coin) is perfect.

□

This is not a trivial theorem. It states that knowledge of both the probabilistic distribution of $m$ and of the cipher $c$, cannot be leveraged for a better estimate of what $m$ actually was.

*Proof.* Formalization in the general setting, using parameters for message distribution instead of 1/3 and 2/3 as we have done—easy.

□

As we have seen getting true random numbers (not pseudo-random!) is important. Some sources you may want to explore if you ever need random numbers (I provide no guarantee!): http://en.wikipedia.org/wiki/Random_number_generator, http://www.random.org/, http://www.elmwoodmagic.com/full/Magic-Tricks-Magic-Books-Magic-DVDs-Red-Casino-Dice-Pack-of-4__3222.htm, http://www.casinochips3000.com/dice2.php.

## 5.4 Entropy (information theory, not physics)

The discussion will be very simplified, focusing on the concept and the intuition.

We randomly pick $x \in X$. How much information about this $x$ is gained by looking at it. (Complementarily, we could say, how much did we know about it, before actually looking at it.)

Note, that if we toss $C_2$, we do not gain anything by looking at the result, because it is always H.

Let us now toss $C_1$ 100 times. There will be some positions (between 0 and 100 in number) in which the result will be H, and there will be some positions (between 0 and 100 in number) in which the result will be T. We want to code the sequence of results using a bit string.

Here is one way of doing it:

1. If there are zero H's, we write 0000000

2. If there is at least one H, we will devote 7 bits to specifying each position in which H was obtained. If we number the tosses from 1 to 100, then each such number can be specified using 7 bits.

**Example 3.** Assume we got H in toss number 17 and toss number 80, and T in all other tosses. Since 17 in binary (using 7 bits) is 0010001 and 80 in binary is 1010000, this sequence of tosses is coded as 00100011010000.

Of course, it would have been more efficient to just use the single bit 0 to denote no heads.

So what is the expected number of bits needed for an answer, it is essentially:

$$7 \cdot \Pr[\text{we got 0 heads}] + 7 \cdot \Pr[\text{we got 1 head}] + 14 \cdot \Pr[\text{we got 2 heads}] + \cdots$$

So, very informally speaking, we get "on the average" that we need "a few bits" to specify the answer—let us say 9, without bothering to compute the actual number here. The first two results happen relatively frequently (we do not compute here how frequently) but the following ones are becoming less and less frequent, so long sequences are unlikely. Of course, in the worst case, the probability of which is tiny, $0.01^{100}$, we will need 700 bits.[40]

So a sequence of 100 tosses "reveals" about 9 bits of "new" information. So each toss reveals about 0.09 bits, if we "average" this out.

But to actually know the result of a single toss, we need 1 bit (H or T). So, in what sense can we talk about a single toss providing less than 1 bit of information. Again, let's go to the intuition.

Alice and Bob play a (one-sided) game. Alice tosses a coin. Bob guesses the result. If he guesses correctly, Alice gives him \$100. If he guesses incorrectly, he gets nothing. Bob knows which coin is used.

Eve sees the result before Bob guesses and offers to sell him the result, so he can guess the answer correctly. How much should Bob pay Eve to increase the amount of money he will actually make.

If $C = C_0$, then no matter what Bob says, he will win (on the average) in half the times, so he will win, on the average \$50 after each toss. So if he pays Eve any amount below \$50, he makes more money net. If Bob pays Eve \$49, Alice pays Bob\$100 and he will win exactly \$51 on each toss.

If $C = C_1$, then if Bob always says H, he will win in 99 out of 100 tosses (on the average). So he benefits by paying Eve only if it is less than \$1. Therefore, knowing the result of the toss is very small (relatively speaking).

To capture the value of obtained information, Shannon `http://en.wikipedia.org/wiki/Claude_Shannon` introduced the concept of entropy `http://en.wikipedia.org/wiki/Information_entropy#Entropy_as_information_content`.

Under very natural conditions, the value of knowing which $x \in X$ actually happened is

$$(10) \quad \mathcal{H}(\mathcal{X}) = \sum_{x \in X} \Pr[x] \cdot \log_2 \left( \frac{1}{\Pr[x]} \right)$$

$H$ is called *entropy* of $\mathcal{X}$. Of course, $\log \frac{1}{y} = \log 1 - \log y = 0 - \log y = -\log y$. Therefore, Eq. 10, is frequently written (easier to typeset and takes

[40]Of course, we make no claim that our coding of the result is optimal—i.e, uses the shortest expected length.

less space):

(11) $\mathcal{H}(\mathcal{X}) = -\sum_{x \in X} \Pr[x] \cdot \log_2 \Pr[x]$

Let us consider one toss of $C_0$. Computing, we get:

$$H = \overbrace{\frac{1}{2}\log_2\frac{2}{1}}^{\text{heads}} + \overbrace{\frac{1}{2}\log_2\frac{2}{1}}^{\text{tails}} = \frac{1}{2} + \frac{1}{2} = 1$$

Intuitively, knowing the toss result is knowing the difference between two outcomes, 1 bit of information.

Let us consider one toss of $C_1$. Computing, we get:

$$H = \overbrace{\frac{99}{100}\log_2\frac{100}{99}}^{\text{heads}} + \overbrace{\frac{1}{100}\log_2\frac{100}{1}}^{\text{tails}} \approxeq 0.08$$

**Note:** To compute $\log_2$, if you can only get $\log_{10}$ from your calculator, use:

(12) $\log_2 z = \frac{\log_{10} z}{\log_{10} 2}$

This follows from $z = 2^{\log_2 z}$, taking $\log_{10}$ from both sides. There is also a free calculator with a "log2" key http://www.bestsoftware4download.com/software/t-free-esbcalc-freeware-calculator-download-hisqvxad.html.

$\square$

Sometimes (not as natural for us) log to a different base is used, which just changes multiplicative constants. The latter follows from Eq. 12.

You can also read about entropy in Forouzan pp. 615–620. We have proved that one-time pad is a perfect (Forouzan misspells as "prefect") in a simpler way, showing all steps. Also, note that he says in Example F.5 that $0 \cdot \log_2(1/0) = 0$. It is OK to say this, but more explanation is needed because $\log_2(1/0)$ is not defined as $1/0$ is infinite.

**Example 4.** Let $D_1$ be a die in which "1" comes up with probability 0.95, and every other number with probability 0.01.

Alice tosses $D_1$. Bob asks her yes/no questions to find out what the result is.

Bob uses the algorithm described in Fig. 34. It is easy to see that the expected number of questions to ask is 1.1 and the entropy is 0.4 (one decimal point of accuracy).

□

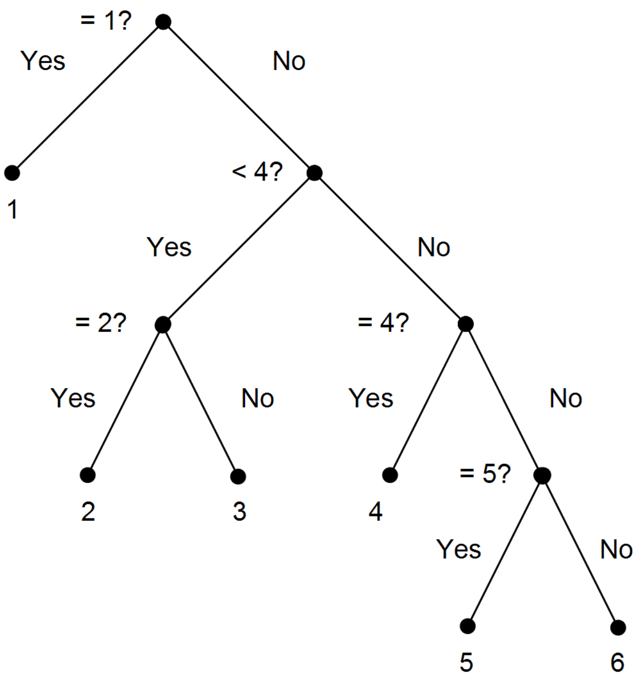


Figure 34: The algorithm Bob uses for determining the result of tossing $D_1$.

There is an important theorem (we state a little informally), which we will not prove, which highlights the importance of entropy and its meaning.

**Theorem 4.** Let $\mathcal{X}$ be a random variable (that is, we get $x_1$ with probability $p_1$, we get $x_2$ with probability $p_2$, …, etc.).

Then if you ask yes/no questions in "an optimal way," the expected number of questions lies between $H(\mathcal{X})$ and $H(\mathcal{X}) + 1$.

(Of course, if $X$ contains at least two elements, you have to ask at least one question.)

□

There is a popular game, called "Twenty Questions" http://en.wikipedia.org/wiki/Twenty_questions. Let us say the "guesser" always guesses in exactly 20 questions. What is the entropy of this game?

An interesting read: `http://danielwilkerson.com/entropy.html`.

## 5.5 Key ideas

1. Probability
2. Conditional probability
3. Bayes theorem
4. Security of the one-time pad cryptosystem
5. Entropy

# 6 Properties of some integers etc.

**Question.** Class discussion (deferred for later, think about this): what is an imaginary number and why?

I will follow Forouzan, as this is very standard material. I may correct, clarify, provide additional/different examples and provide more and/or different justifications/proofs. So, all pages from Forouzan that I list are "incorporated" in these notes, without of course copying them, as this might violate copyright.

## 6.1 Operations on integers

### Material covered

Forouzan, pp. 24–27.

### Comment for p. 20; notation only

Forouzan uses **Z** to denote set(s) of integers. The more common notation is $\mathbb{Z}$.

### Comment for p. 24

Here, and elsewhere, although Forouzan provides proofs in Appendix Q, I would like to give you somewhat different proofs, which I believe are simpler.

**Theorem 5.** The Euclidean algorithm works.

□

*Proof.* Note that we have $r_1 > r_2 > 0$.

Instead a complete proof by induction, we will just do the first stage.

Let us rewrite these first two stages as:[41]

1. $r_1 = q_1 r_2 + r_3$ with $0 \le r_3 < r_2$
2. $r_2 = q_2 r_3 + r_4$ with $0 \le r_4 < r_3$

[41] There are various conventions for numbering. Following, Forouzan, I am using $r_1$ and $r_2$. Some people start with $r_0$ and write $r_0 = q_1 r_1 + r_2$. Of course, it is of no importance what is chosen as long as then everything is done consistently.

In general, we could write: $r_i = q_i r_{i+1} + r_{i+2}$, starting with $i = 1$ and ending when $r_{i+2} = 0$. Then $r_{i+1}$ will be $\gcd(r_1, r_2)$.

We will show that we are "zeroing on" the greatest common divisor, because $\gcd(r_1, r_2) = \gcd(r_2, r_3)$ (which we will prove). So we reduce a larger problem to a smaller problem.

To do this, it is enough to show that (for every) $d$,[42]

$$r_1 \text{ and } r_2 \text{ divisible by } d$$

iff

$$r_2 \text{ and } r_3 \text{ divisible by } d$$

We will be using various Greek letters to indicate some expressions in the way that is obvious from the context.

1. Let $r_1 = \alpha d$, $r_2 = \beta d$, for some $\alpha$ and $\beta$.

   We need to show that $r_2$ and $r_3$ are divisible by $d$. We already know that $r_2$ is divisible by $d$. $r_3 = r_1 - q_1 r_2 = \alpha d - q_1 \beta d = (\alpha - q_1 \beta) d = \gamma d$, and therefore is divisible by $d$.

2. Let $r_2 = \alpha d$, $r_3 = \beta d$, for some $\alpha$ and $\beta$.

   We need to show that $r_1$ and $r_2$ are divisible by $d$. We already know that $r_2$ is divisible by $d$. $r_1 = q_1 r_2 + r_3 = q_1 \alpha d + \beta d = (q_1 \alpha + \beta) d = \gamma d$, and therefore is divisible by $d$.

□

## Comment for p. 26

**Theorem 6.** The extended Euclidean algorithm works.

□

*Proof.* This is a little more complicated, so I will provide a complete proof by induction. To do this, we will not "reuse" $r_1$, $r_2$, $q_1$, $q_2$, $s_1$, $s_2$, $t_1$, $t_2$, but give new indices to new values.

So we define $q_1, q_2, q_3, \ldots; r_1, r_2, r_3, \ldots; s_1, s_2, s_3, \ldots; t_1, t_2, t_3, \ldots$.

[42] Forouzan writes $d \mid x$, read "$d$ is a divisor of $x$" or "$d$ divides $x$," which of course means that $x$ is divisible by $d$. He uses what is a standard notation, but I will frequently spell this out as I feel this is easier to read.

We have, as done in the Euclidean algorithm:

(13) $r_i = q_i r_{i+1} + r_{i+2}$

We define

(14) $s_j = \begin{cases} 1 & j = 1 \\ 0 & j = 2 \\ s_{j-2} - q_{j-1}s_{j-1} & j > 2 \end{cases}$

(15) $t_j = \begin{cases} 0 & j = 1 \\ 1 & j = 2 \\ t_{j-2} - q_{j-1}t_{j-1} & j > 2 \end{cases}$

and we will show that

(16) $r_j = s_j r_1 + t_j r_2$ for all $j \geq 1$ until the algorithm terminates

We proceed by induction:

1. $j = 1$. From the way we defined the various sequences, we see that, indeed, $r_1 = s_1 r_1 + t_1 r_2$.

2. $j = 2$. From the way we defined the various sequences, we see that, indeed, $r_2 = s_2 r_1 + t_2 r_2$.

3. $j > 2$.

   Recall that: $r_i = q_i r_{i+1} + r_{i+2}$, for $i \geq 1$. We can, of course, rewrite this, by writing $j - 2$ for $i$, as:

   $$r_{j-2} = q_{j-2}r_{j-1} + r_j \text{ because as } j \geq 3 \text{ we have that } j - 2 \geq 1$$

   Therefore by rewriting and also using induction to substitute for $r_{j-1}$ and $r_j$, we have:

$$\begin{aligned}
r_j &= r_{j-2} - q_{j-2}r_{j-1} \text{ (rewriting)}\\
&= (s_{j-2}r_1 + t_{j-2}r_2) - q_{j-2}(s_{j-1}r_1 + t_{j-1}r_2) \text{ (by induction)}\\
&= (s_{j-2} - q_{j-2}s_{j-1})r_1 + (t_{j-2} - q_{j-2}t_{j-1})r_2 \text{ (rewriting)}\\
&= s_j r_1 + t_j r_2 \text{ (by definition of } s\text{'s and } t\text{'s)}
\end{aligned}$$

□

## 6.2 Modular arithmetic

Forouzan, pp. 29–40.

### Comment for p. 33

Although Forouzan provides proofs in Appendix Q, I would like to give you somewhat different proofs, which I believe are simpler.

There are three properties. We will prove the third one (the other ones are even simpler to prove).

**Theorem 7.** (This is so simple that it should not really be called a theorem.)

$$(a \times b) \bmod n = [(a \bmod n) \times (b \bmod n)] \bmod n$$

*Proof.* Let

$$\begin{aligned}
a &= \alpha n + \gamma \qquad \text{with } 0 \le \alpha \text{ and } 0 \le \gamma < n\\
b &= \beta n + \delta \qquad \text{with } 0 \le \beta \text{ and } 0 \le \delta < n
\end{aligned}$$

Let us start with the left-hand side of the equation to be proved.

$$a \times b = \alpha\beta n^2 + (\alpha\delta + \beta\gamma)n + \gamma\delta$$

We can write:

$$\gamma\delta = \epsilon n + \zeta \qquad \text{with } 0 \le \epsilon \text{ and } 0 \le \zeta < n$$

Then,

$$(a \times b) \bmod n = \zeta$$

Let us compute now the right-hand side of the equation to be proved:

$$[(a \bmod n) \times (b \bmod n)] = \gamma\delta$$

We have

$$a \bmod n = \gamma$$

and

$$b \bmod n = \delta$$

Therefore,

$$[(a \bmod n) \times (b \bmod n)] = \gamma\delta = \epsilon n + \zeta \text{ as before}$$

and therefore, as before;

$$[(a \bmod n) \times (b \bmod n)] \bmod n = \zeta$$

□

### Elaboration on p. 36

Finding multiplicative inverses is very important for us, so let us go through an example very carefully in all its stages. See Fig. 35.

| q | $r_1$ | $r_2$ | r | $s_1$ | $s_2$ | s | $t_1$ | $t_2$ | t |
|---|---|---|---|---|---|---|---|---|---|
| 2 | **75** | **28** | 19 | **1** | **0** | 1 | **0** | **1** | -2 |
| 1 | 28 | 19 | 9 | 0 | 1 | −1 | 1 | -2 | 3 |
| 2 | 19 | 9 | 1 | 1 | −1 | 3 | −2 | 3 | -8 |
| 9 | 9 | 1 | 0 | −1 | 3 | -28 | 3 | -8 | 75 |
|  | **1** | 0 |  | **3** | -28 |  | **-8** | 75 |  |

Figure 35: Extended Euclidean algorithm for 75 and 28.

We want to find the multiplicative inverse of 28 modulo 75, if such multiplicative inverse exists.

1. We run the Extended Euclidean algorithm and we see that $\gcd(75, 28) = 1$ (for this we only needed the Euclidean algorithm, but we need the Extended Euclidean algorithm for the rest so may as well run it now). Therefore we know that the desired multiplicative inverse exists.

2. We look at the one before last row of the table, which is really when the algorithm is finished. From it we deduce that:

   $$1 = 3 \cdot 75 + (-8) \cdot 28$$

   and $-8$ is a multiplicative inverse of 28 modulo 75.

3. We do not like negative numbers, as everything should be in the range $\{1, 2, \ldots, 74\}$. We would like to put $-8$ into this range, very easy we add 75 to it.[43] But of course, the equation has to remain correct. So what we do is:

   $$1 = 3 \cdot 75 + (-8 + 75) \cdot 28 - 75 \cdot 28$$

   So, we really added to the right side: $+75 \cdot 28 - 75 \cdot 28 = 0$ Rewriting,

   $$1 - 3 \cdot 75 + 28 \cdot 75 = 67 \cdot 28$$

   or,

   $$1 + 25 \cdot 75 = 67 \cdot 28$$

   or,

   $$67 \cdot 28 = 25 \cdot 75 + 1$$

Therefore, $67 \cdot 28 = 1 \bmod 75$, and therefore 67 is a multiplicative inverse of 28 modulo 75.[44]

### Comment for p. 39

Forouzan uses the notation $\mathbf{Z}_{\mathbf{n}^*}$. The asterisk is in the wrong place. It should be superscript of "the whole thing" not of $n$. So it should be $\mathbf{Z}_n^*$, or actually $\mathbb{Z}_n^*$.

[43] Generally we can put any integer into the range but adding or subtracting some multiple of 75.

[44] Yes, it is really true that $\frac{1}{28} = 67$ when computed modulo 75.

**Addition to p. 39**

It is really a good idea to produce multiplication table for $\mathbb{Z}_{10}^*$. It is shown in Fig. 36

| $\times_{10}$ | 1 | 3 | 7 | 9 |
|---|---|---|---|---|
| 1 | 1 | 3 | 7 | 9 |
| 3 | 3 | 9 | 1 | 7 |
| 7 | 7 | 1 | 9 | 3 |
| 9 | 9 | 7 | 3 | 1 |

Figure 36: Multiplication table for $\mathbb{Z}_{10}^*$. $\times_{10}$ stands for multiplication modulo 10. Similar notation is used later too.

Note that you can easily read out of it various multiplicative inverses. For instance, the multiplicative inverse of 7 is 3.

**Addition to p. 39**

Let us consider multiplication table for $\mathbb{Z}_7^*$. It is shown in Fig. 37

| $\times_7$ | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 3 | 4 | 5 | 6 |
| 2 | 2 | 4 | 6 | 1 | 3 | 5 |
| 3 | 3 | 6 | 2 | 5 | 1 | 4 |
| 4 | 4 | 1 | 5 | 2 | 6 | 3 |
| 5 | 5 | 3 | 1 | 6 | 4 | 2 |
| 6 | 6 | 5 | 4 | 3 | 2 | 1 |

Figure 37: Multiplication table for $\mathbb{Z}_7^*$.

**Question.** I have three friends and I want to buy six cookies and divide equally among them. Each cookie costs \$2. How much money I am going to spend on each of my friends?

$\square$

Let us look at even bigger example, that of $\mathbb{Z}_{15}^*$ on Fig. 38.

There is something that Forouzan (at least partially) glosses over, but which is of tremendous importance, so we will even call it a theorem.

**Theorem 8.** Two claims:

| $\times_{15}$ | 1 | 2 | 4 | 7 | 8 | 11 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 4 | 7 | 8 | 11 | 13 | 14 |
| 2 | 2 | 4 | 8 | 14 | 1 | 7 | 11 | 13 |
| 4 | 4 | 8 | 1 | 13 | 2 | 14 | 7 | 11 |
| 7 | 7 | 14 | 13 | 4 | 11 | 2 | 1 | 8 |
| 8 | 8 | 1 | 2 | 11 | 4 | 13 | 14 | 7 |
| 11 | 11 | 7 | 14 | 2 | 13 | 1 | 8 | 4 |
| 13 | 13 | 11 | 7 | 1 | 14 | 8 | 4 | 2 |
| 14 | 14 | 13 | 11 | 8 | 7 | 14 | 2 | 1 |

Figure 38: Multiplication table for $\mathbb{Z}_{15}^*$.

1. If $a \in \mathbb{Z}_n^*$, then for some $b \in \mathbb{Z}_n^*$, we have $a \times_n b = 1$. This really means that $a$ has a multiplicative inverse, $b$, inside $\mathbb{Z}_n^*$.

2. If $a \in \mathbb{Z}_n^*$, and $b \in \mathbb{Z}_n^*$, then $a \times_n b \in \mathbb{Z}_n^*$

$\square$

*Proof.*

1. If $a \in \mathbb{Z}_n^*$, then for some $b \in \mathbb{Z}_n^*$, we have $a \times_n b = 1$

   Pick any such $a$. By definition of $\mathbb{Z}_n^*$ it is relatively prime with $n$.[45] So there is a multiplicative inverse modulo $n$ (by Extended Euclidean algorithm). Let it be $b$. We need to show that it is in $\mathbb{Z}_n^*$.

   We will show that $b$ is relatively prime with $n$. Assume otherwise, and let us say that they share some factor $d > 1$. Then, $b = \alpha d$ and $n = \beta d$. We have, by definition of "multiplicative inverse" that

$$ab = \gamma n + 1 \qquad \text{for some } \gamma$$

we had q instead of 1 by mistake

   and therefore

$$a\alpha d = \gamma\beta d + 1 \qquad \text{and} \qquad (a\alpha - \gamma\beta)d = 1$$

   which is impossible as the left hand side is divisible by $d$ and the right hand side is not.

[45] *relatively prime* and *coprime* mean the same thing

2. If $a \in \mathbb{Z}_n^*$, and $b \in \mathbb{Z}_n^*$, then $a \times_n b \in \mathbb{Z}_n^*$

   What we really need to show is that $a \times b \bmod n$ is relatively prime with $n$ (and therefore in $\mathbb{Z}_n^*$).

   Let

$$a \times b = c + \alpha n \qquad \text{for some } c \text{ and } \alpha$$

   Assume that $c$ is not relatively prime with $n$ and they do share some factor $d > 1$. Then we can write

$$c = \beta d \qquad \text{and} \qquad n = \gamma d$$

   Therefore,

$$a \times b = \beta d + \alpha\gamma d = (\beta + \alpha\gamma)d$$

   So either $a$ or $b$ (or both) is divisible by $d$ and therefore shares a factor with $n$ (which is divisible by $d$). But this is impossible as $a$ and $b$ are relatively prime with $n$, and therefore do not share factors with it.

□

## 6.3 Key ideas

1. Euclidean algorithm and Extended Euclidean algorithm for:
   (a) Finding greatest common divisor
       Finding a multiplicative inverse modulo
2. Modular arithmetic
   (a) Modular addition and $\mathbb{Z}_n$
   (b) Modular multiplication and $\mathbb{Z}_n^*$

# 7 RSA public/private encryption system

## 7.1 Preliminaries

Again, I will follow Forouzan as closely as possible. I will "strip out" what is not needed, may add more examples, and mathematical justifications. I will be very specific about what is taken from Forouzan, by listing various "locations" in the book using our [page number, line number] notation. To save space, I will actually write F[page number, line number].

First, whenever

$$a = \alpha n + \gamma \qquad \text{and} \qquad b = \beta n + \gamma$$

that is when $a$ and $b$ are equal modulo $n$, we may say that they are *congruent* modulo $n$ and write:

$$a \equiv b \bmod n$$

This is standard notation, which we have not used before, see F[30, −7].[46]

**F[251, −7]–F[253, −10]**

We skip his discussion on primality testing, we need to do something else which we (and he) will do later.

**F[254, −8]–F[255, −2]**

**Definition 4.** For any interger $n \geq 1$, $\phi(n)$ will denote the number of of positive integers smaller than $n$ that are relatively prime with $n$. Note that this is the cardinality (size) of the set $\mathbb{Z}_n^*$.

□

**Example 5.** $\phi(10) = 4$, since the following are all the relevant integers: 1, 3, 7, 9.

□

We do not need to know all that F tells us. We only need to know that (and prove it):

**Theorem 9.** Two claims:

1. If $p$ is prime, then $\phi(p) = p - 1$

[46]We will not write, unless there is a special reason, "(" and ")" as Forouzan does.

2. If $n = pq$, and $p$ and $q$ are primes, the $\phi(n) = (p-1)(q-1)$

□

*Proof.* Instead of dragging in indices/variables, etc., our proofs will be by "general examples."[47]

1. If $p$ is prime, then $\phi(p) = p - 1$

   Pick $p = 7$. Then the numbers relative prime with 7 are 1, 2, 3, 4, 5, 6; i.e., all the positive integers smaller than 7, of which there are 6.

2. If $n = pq$, and $p$ and $q$ are primes, the $\phi(n) = (p-1)(q-1)$

   Pick $n = 15$. Then $n = 3 \cdot 5$, so $p = 3$ and $q = 5$. Let us consider all the candidates for being included in the set of integers smaller than 15 that are relatively prime with 15. The set to look at is 1, 2, ..., 14.

   Let us look for those numbers that *are not relatively prime* with 15. They are either multiples of 3 or or 5 (cannot be of both as we are looking at numbers smaller than 15). It will be convenient to actually look at the set 1, 2, ..., 14, 15.

   (a) Let us consider multiples of 3 (which is $p$).

   They are 3, 6, 9, 12, 15. How many of them are: $\frac{15}{3} = 5$ (simply because we have "gaps" of 3, and $3 \cdot 5 = 15$). Note that $5 = q$

   But 15, does not count as being of interest as it is too big. Therefore there are only $4 = 5 - 1 = q - 1$ such numbers.

   (b) Let us consider multiples of 5 (which is $q$).

   They are 5, 10, 15. How many of them are: $\frac{15}{5} = 3$ (simply because we have "gaps" of 5, and $5 \cdot 3 = 15$). Note that $5 = p$

   But 15, does not count as being of interest as it is too big. Therefore there are only $2 = 3 - 1 = p - 1$ such numbers.

   There were 14 candidates, but $4 + 2$ did not work out. So there are 8 left. Note that $8 = (5-1)(3-1)$.

   This perhaps still looks mysterious, so let us now write, using what we know:

[47] I try to do it as much as possible in cases where the difference between such a proof and and a general proof is just writing variables instead of specific examples, so we have more intuition and don't really lose any generality.

$$\phi(pq) = \overbrace{(pq-1)}^{14} - \overbrace{(p-1)}^{4} - \overbrace{(q-1)}^{2} = pq - p - q + 1 = (p-1)(q-1)$$

□

Let us now look at $\mathbb{Z}^*_{15}$. As we have a table of multiplication for it, Fig 38, we can produce powers of various elements of it, $a^1 \bmod 15 = a, a^2 \bmod 15, a^3 \bmod 15, \ldots$. Let us do it.

We get:
1, 1, 1, 1, 1, 1, 1, 1, ...
2, 4, 8, 1, 2, 4, 8, 1, ...
4, 1, 4, 1, 4, 1, 4, 1, ...
7, 4, 13, 1, 7, 4, 13, 1, ...
8, 4, 2, 1, 8, 4, 2, 1, ...
11, 1, 11, 1, 11, 1, 11, 1, ...
13, 4, 7, 1, 13, 4, 7, 1, ...
14, 1, 14, 1, 14, 1, 14, 1, ...

Note that for any $a$,[48]

(17) $a^8 \bmod 15 = 1$

Let us now look at number 3. It is relatively prime with 8. So there is a multiplicative inverse modulo 8. And it is easily found using Extended Euclidean algorithm: 11.[49]

Just to check:

$$3 \times 11 \bmod 8 = 33 \bmod 8 = (4 \times 8 + 1) \bmod 8 = 1$$

For us, the important things is that since 3 and 11 are multiplicative inverses modulo 8, we can write, for some $\alpha$ (which happens to be 4, which is not important) that

$$3 \times 11 = \alpha \times 8 + 1 \qquad \text{and in this case} \qquad 3 \times 11 = 4 \times 8 + 1$$

[48]The real reason for this is that $8 = \phi(15)$ so we can also write $a^{\phi(15)} \bmod 15 = 1$, but we will return to this later.

[49]We are cheating a little. We will discuss it soon.

From this it follows that

(18) $a^{33} \bmod 15 = a$

Indeed,

$$\begin{aligned} a^{33} \bmod 15 &= a^{32+1} \bmod 15 \\ &= a^{8^4} \times a^1 \bmod 15 \\ &= \big([(a^8 \bmod 15)^4 \bmod 15] \times [a \bmod 15]\big) \bmod 15 \\ &= [1^4 \bmod 15] \times a \qquad \text{(by Eq. 17)} \\ &= a \end{aligned}$$

From this we have a special (and incomplete for the time being) case of RSA—but all the key ideas are here!

**Example 6.** Alice picks as her public key the pair $(15, 3)$ and as her private key the pair $(15, 11)$. Both keys are consist of a pair of the form (modulus, exponent).

Bob wants to send $x = 8$ to Alice.

Bob computes $y = x^3 \bmod 15 = 8^3 \bmod 15 = 2$ and sends it to Alice.[50]

Alice computes $z = y^{11} \bmod 15 = 2^{11} \bmod 15 = 8$.[51]

And

$$z = x$$

But this was *guaranteed* by Eq. 18.

□

**Note** The example was in some sense fake. Actually, as multiplicative inverse of 3 was computed modulo 8, even though 11 is correct, we want multiplicative inverse that is $< 8$. Why, it is secret and we need to worry about its being guessed. So we want to see how big it is. And the smallest possible multiplicative inverse will be the unique one that is $< 8$. And in our example it is actually 3. So 3 is its own multiplicative inverse. In practice this will not happen, but our example was very tiny on purpose so everything could be computed easily, and therefore it did happen in it. So let us take

[50] In fact, if you compute: $8^3 = 2 + 34 \times 15$.

[51] In fact, if you compute: $2^{11} = 8 + 136 \times 15$.

the example in Forouzan that is bigger (but therefore more tedious to check out). Before looking at it, we just need to believe, which we will prove later (see Theorem 11, that if $n = p \times q$, where $p$ and $q$ are prime, and for any $a$, s.t. $0 \leq a < n$, and any $k \geq 1$, the following holds: $a^{k \times \phi(n)+1} \equiv a \bmod n$

**F[304]–F[305]**[52] There are several examples there. I will list one of the explicitly below.

**F[312]–F[314]** This is a really big example. No realistic hope of checking this out unless you have software for manipulating large integers precisely.

**Example 7.** F[304] We exchange the roles of Alice and Bob. (Alice always goes first and the first person is the one that chooses the keys.)

$n = 77 = 7 \times 11.$

$\phi(n) = (7-1) \times (11-1) = 60.$

$e = 13$ is relatively prime with 60 and Alice chooses $(77, 13)$ as her public key.

She computes a multiplicative inverse of 13 modulo 60 and gets 37. Alice chooses $(77, 37)$ as her private key.

Bob wants to send $x = 5$ to Alice.

Bob computes $y = x^{13} \bmod 77 = 26$ and sends it to Alice.

Alice computes $z = y^{37} \bmod 77 = 5$.

And

$$z = x$$

□

## 7.2 The RSA algorithm

**F[303]**

This is the RSA algorithm.[53]

A couple of comments:

1. I would say that the private key is the pair $(d, n)$ and not just $d$. Of course I am not implying anything really different than what F says. It is just that you really need to know both $d$ and $n$ for decryption and perhaps you decided you do not need to ever encrypt so you do not remember/know what the pair $(e, n)$ is.

[52] When line numbers are clear from the text, I do not list them

[53] We can also look now at page 83.

2. At this point, we do not need to know what Fast Exponentiation is. It pertains to efficiency as opposed to correctness. We will discuss it later

3. This actually pertains to F[302]. We do not need to know this. But just for the record, the notation for the ring as written by F uses standard "greater than" and "less than" symbols, so it looks like $< \ldots >$. It really should be a different symbol and look like $\langle \ldots \rangle$. These are, so called, "angle brackets."

We now need to show this works.

Theorem 10 essentially encompasses Euler's theorem, first version as written in F[257]. Fermat theorem, first version,[54] as written in F[256] is a special case.[55] F[684] sketches a proof, which I think is a little too sketchy. I provide a complete proof.

**Theorem 10.** For every $n$ and $a \in \mathbb{Z}_n^*$,[56] we have that

$$a^{\phi(n)} \equiv 1 \bmod n$$

□

*Proof.* Here and in some other places we will be very explicitly about the operation used. I will not write $xy$, but either $x \times y$ or $x \times_n y$, so we are not confused about the operation and note make any mistakes.

We will write $f$ for $\phi(n)$, just to make the formulas shorter to write.

Let all the elements of $\mathbb{Z}_n^*$ be: $a_1, a_2, \ldots, a_f$. Let us assume that there are listed in increasing order, and therefore $1 = a_1 < a_2 < \cdots < a_f < n$.

We first show that all of the below are different elements of $\mathbb{Z}_n^*$:

$a \times_n a_1$
$a \times_n a_2$
$\ldots$
$\ldots$
$a \times_n a_f$

First we note that as by Theorem 8, $\mathbb{Z}_n^*$ is closed under $\times_n$, these are elements of $\mathbb{Z}_n^*$. We now show they are different.

Assume by contradiction that for some $i < j$, we have

[54]Frequently also called Fermat's little theorem.

[55]Recall that if $p$ is prime, then $\phi(p) = p - 1$.

[56]Forouzan states the theorem somewhat more generally, by allowing $a$ to be larger than $n$ as long as they are relatively prime, we do not need to discuss this here—the proof will work exactly the same with extremely minor changes.

$$a \times_n a_i = a \times_n a_j$$

Then we have:

$$a \times a_i = \alpha + \beta n \qquad \text{and} \qquad a \times a_j = \alpha + \gamma n \qquad \text{for some} \qquad \beta < \gamma$$

And therefore:

$$a \times (a_j - a_i) = (\gamma - \beta) \times n \qquad \text{and this is} > 0 \text{ as } a_j > a_i$$

As the right hand side is divisible by $n$, the left hand side is divisible by $n$ also. But, as $a$ and $n$ are relatively prime, they do not share any factors, therefore for $a \times (a_j - a_i)$ to be divisible by $n$, $(a_j - a_i)$ must be divisible by $n$. But as $0 < a_i < a_j < n$, we have $0 < a_j - a_i < n$, and therefore this is not possible.

> If we look at our example for $n = 15$, if we have $a = 8$, then we have $8 \times (a_j - a_i) = (\gamma - \beta) \times 15$. Therefore $(a_j - a_i)$ must be divisible by 15, which is impossible, as it is positive but smaller than 15.

So we have proved our claim that all the $a \times_n a_i$'s are different elements of $\mathbb{Z}_n^*$

So we can write for each $i$:

(19) $a \times a_i = \alpha_i + \beta_i \times n$

for some $\alpha_i$ and $\beta_i$. Here $\alpha_i = a \times_n a_i$.

Furthermore:

$\alpha_1$
$\alpha_2$
...
...
$\alpha_f$

are all different elements of $\mathbb{Z}_n^*$.[57]

[57]We are not saying that $\alpha_1 = a_1$, etc.; we are saying that if we look at all the $\alpha_i$'s, we get all the $a_i$'s.

We define three quantities:

(20) $A = (a \times a_1) \times (a \times a_2) \times \cdots \times (a \times a_f)$

(21) $B = a_1 \times a_2 \times \cdots \times a_f$

(22) $C = \overbrace{(a \times a \times \cdots \times a)}^{f \text{ times}}$ Note that $C = a^{\phi(n)}$

We have:

$$A = \overbrace{(a \times a \times \cdots \times a)}^{f \text{ times}} \times (a_1 \times a_2 \times \cdots \times a_f) = C \times B$$

But using Eq. 19, we can also write

$$\begin{aligned} A &= (\alpha_1 + \beta_1 \times n) \times (\alpha_2 + \beta_2 \times n) \times \cdots \times (\alpha_f + \beta_f \times n) \\ &= \alpha_1 \times \alpha_2 \times \cdots \times \alpha_f + \gamma \times n \qquad \text{for some } \gamma \\ &= a_1 \times a_2 \times \cdots \times a_f + \gamma \times n \qquad \text{because each } \alpha_i \text{ is some } a_j \\ &= B + \gamma \times n \end{aligned}$$

So we, get

(23) $C \times B = B + \gamma \times n$

We can write, for some $g \in \mathbb{Z}_n$ and some $\delta$:

$$C = g + \delta \times n$$

But, as $C$ is a product of some number of $a$'s (actually, of course, $n$ of them) and $a$ is relatively prime with $n$, it follows that $g$ is relatively prime with $n$, which means that $g \in \mathbb{Z}_n^*$.

By Eq. 23, we have

$$(g + \delta \times n) \times B = B + \gamma \times n$$

and therefore

$$(g-1) \times B = (\gamma - \delta \times B) \times n$$

The right hand side is divisible by $n$ and therefore the left hand side must be divisible by $n$ also.

But we know that as $B$ is a product of integers all relatively prime with $n$, $B$ cannot share any factors with $n$. Therefore, $g-1$ has to be divisible by $n$. But as $g < n$, this is only possible when $g-1=0$.

We have shown that $g=1$, and therefore

$$C = 1 + \delta \times n$$

In other words:

$$a^{\phi(n)} \equiv 1 \bmod n$$

□

**Note** We have not discussed formally what happens when we want to encrypt $a$ in $Z_n$, which is not in $\mathbb{Z}_n^*$, we will show that "things work" soon, but meanwhile let's have a digression.

Here we look how everything was set up for RSA to work, following the example we had from Forouzan:[58]

1. Alice picks two large primes randomly, say $p$ and $q$. Easy, using Miller-Rabin algorithm, which we will cover soon.

   (We had $p = 7$ and $q = 11$.)

2. Let $n = p \times q$. Alice computes $\phi(n) = (p-1) \times (q-1)$. Trivial. Note that $\phi(n)$ is large, as it is almost as big as $p \times q$.

   (We had $n = 7 \times 11 = 77$ and $\phi(77) = (11-1) \times (7-1) = 60.$[59])

3. She picks a small element $e \in \mathbb{Z}_n^*$, s.t. $e$ is relatively prime with $\phi(n)$. Easy, she looks at some small integers and will soon find an integer with the desired property as $\phi(n)$ cannot have too many different prime factors or it would be huge. Testing for relatively primality is easy using the Euclidean algorithm.

[58] In real implementations things are done somewhat differently, but this is the core of how it is done.

[59] Why does an hour have 60 minutes?

(We had $e = 13$.)[60]

4. Alice computes $d$, the multiplicative inverse of $e$ modulo $\phi(n)$. Easy, using Extended Euclidean algorithm. Note that as $e$ was small, $d$ was automatically large, as $e \times d = \alpha \times \phi(n) + 1$ for some $\alpha \geq 1$. The value of $\alpha$ is actually not important.

   (We had $d = 37$ and $e \times d = 13 \times 37 = 481 = 8 \times 60 + 1$.)[61]

5. Alice publishes $n$ and $e$. Together they form $e^{\text{RSA}}_{\text{Alice}}$. Alice keeps $d$ secret. $n$ and $d$ together form $d^{\text{RSA}}_{\text{Alice}}$.

   (We had public 77 and 37 and private 77 and 13.)

6. Bob wants to send a message $m$, $0 \leq m < n$ to Alice. He encrypts: $c = m^e \bmod n$ and sends $c$ to Alice.

   (We had $m = 5$ and $c = 26$.)

7. Alice gets $c$ and decrypts $z = c^d \bmod n$. This will be the original message $m$.

   (We had $c = 26$ and $z = 5$.)

Let us recall that we have shown in Theorem 10, that when if $a \in \mathbb{Z}_n^*$, then $a^{\phi(n)} \equiv 1 \bmod n$. And from this it followed that $a^{k \times \phi(n)+1} \equiv a \bmod n$ and "RSA worked" for such an $a$.

But what about $a$ "outside" $\mathbb{Z}_n^*$, that is $a \in \mathbb{Z}_n \setminus \mathbb{Z}_n^*$? Could we use RSA to encrypt such $a$'s?

In some sense it does not matter as for large $n$, $\mathbb{Z}_n^*$ is "practically all" of $\mathbb{Z}_n$, so we will never in practice wil encounter encryption of elements in $\mathbb{Z}_n \setminus \mathbb{Z}_n^*$.[62].

But let us try and apply RSA to all the elements of $\mathbb{Z}_n$. Let us look at Fig. 39.

Unfortunately for some elements, we indeed have that $a^{\phi(n)} \not\equiv 1 \bmod n$. However what we really need is $a^{k \times \phi(n)+1} \equiv a \bmod n$ (for some specific $k$ actually the one for which $e \times d = k \times \phi(n) + 1$). And, in our example, we indeed see that $a^{1 \times \phi(n)+1} \equiv a \bmod n$ for every $a$. But what about the value

[60]Actually, $e$ should be very small relatively to $n$, which is not the case in this example. We could have chosen $e = 7$ and $d = 43$, but that's not what Forouzan did.

[61]In our example $e$ was not small relative to $n$, but nevertheless we got a large $d$. The example is too small to be able to talk more carefully about the sizes of the keys.

[62]This will need to be made more precise, as we will do later as we encrypt "partially random" numbers in practice.

| $a$ | $a^1$ | $a^2$ | $a^3$ | $a^4$ | $a^5$ | $a^6$ | $a^7$ | $a^8$ | $a^9$ |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 2 | 4 | 8 | 1 | 2 | 4 | 8 | 1 | 2 |
| 3 | 3 | 9 | 12 | 6 | 3 | 9 | 12 | 6 | 3 |
| 4 | 4 | 1 | 4 | 1 | 4 | 1 | 4 | 1 | 4 |
| 5 | 5 | 10 | 5 | 10 | 5 | 10 | 5 | 10 | 5 |
| 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| 7 | 7 | 4 | 13 | 1 | 7 | 4 | 13 | 1 | 7 |
| 8 | 8 | 4 | 2 | 1 | 8 | 4 | 2 | 1 | 8 |
| 9 | 9 | 6 | 9 | 6 | 9 | 6 | 9 | 6 | 9 |
| 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| 11 | 11 | 1 | 11 | 1 | 11 | 1 | 11 | 1 | 11 |
| 12 | 12 | 9 | 3 | 6 | 12 | 9 | 3 | 6 | 12 |
| 13 | 13 | 4 | 7 | 1 | 13 | 4 | 7 | 1 | 13 |
| 14 | 14 | 1 | 14 | 1 | 14 | 1 | 14 | 1 | 14 |

Figure 39: Powers in $\mathbb{Z}_{15}$

of $k$ for which we will need this? We can prove that the equation holds for *every* $k$, including the one we will need.

Theorem 11 is stated in F[257] as the second version of Euler's theorem. He also sketches a proof. Again, I provide a complete proof.

**Theorem 11.** Let $n = p \times q$ for two distinct primes $p$ and $q$. Let $a$, $0 \leq a < n$. Then, for any $k \geq 1$:

(24) $a^{k\times\phi(n)+1} \equiv a \bmod n$

$\square$

*Proof.* Assume first that $a$ is relatively prime with $n$.

By Theorem 10:

$$\begin{aligned} a^{k\times\phi(n)+1} &= a \times (a^{\phi(n)})^k \\ &= a \times (1 + \alpha \times n)^k \qquad \text{By Theorem 10 for some } \alpha \\ &= a \times (1 + \beta \times n) \qquad \text{for some } \beta \\ &= a + \gamma \times n \qquad \text{for some } \gamma \end{aligned}$$

which proves the claim.

Assume now that $a$ is not relatively prime with $n$ $a$ is either a multiple of $p$ or a multiple of $q$ (but not of both, because then $a$ would be a multiple of $n$). Without loss of generality, let's assume that it is a multiple of $p$

We can write (for suitable Greek letters, here and in the rest of the proof): $a = \alpha + \beta \times q$. And as $a$ relatively prime with $q$, $\alpha$ is relatively prime with $q$.

Note that $\phi(n) = \phi(p) \times \phi(q)$, since

1. $\phi(n) = (p-1) \times (q-1)$
2. $\phi(p) = p - 1$
3. $\phi(q) = q - 1$

Of course,

$$a^{k\times\phi(n)+1} = \left((\alpha + \beta \times q)^{\phi(q)}\right)^{\phi(p)\times k} \times a$$

But,

$$(\alpha + \beta \times q)^{\phi(q)} = \alpha^{\phi(q)} + \gamma \times q$$

Also, by Theorem 10,

$$\alpha^{\phi(q)} = 1 + \delta \times q \qquad \text{(since } \alpha \text{ is relatively prime with } q\text{)}$$

Putting it all together, we have:

$$a^{k\times\phi(n)+1} = (1 + \delta \times q + \gamma \times q)^{\phi(p)\times k} \times a = (1 + \epsilon \times q) \times a = a + \zeta \times q \tag{25}$$

But, by assumption, $a = \eta \times p$. Therefore:

$$\begin{aligned} a^{k\times\phi(n)+1} &= a \times a^{k\times\phi(n)} \\ &= a \times 1 + a \times (a^{k\times\phi(n)} - 1) \\ &= a + \theta \times a \\ &= a + \theta \times \eta \times p \\ &= a + \iota \times p \end{aligned}$$

So, from Eq. 25

$$a + \zeta \times q = a + \iota \times p$$

and

$$\zeta \times q = \iota \times p$$

Therefore, $\zeta$ divisible by $p$, and

$$\zeta = \kappa \times p$$

Finally,

$$\begin{aligned} a^{k \times \phi(n)+1} &= a + \zeta \times q \\ &= a + \kappa \times p \times q \\ &= a + \kappa \times n \end{aligned}$$

□

## 7.3 From the RSA algorithm to an RSA protocol

So we have the RSA algorithm. Let us now explore some additional issues that need to be taken care of before we can use it for encryption/decryption and signing.

The message to encrypted is $m$ and the resulting cipher is $c$.

### Attempt to break: factoring of $n$

Eve sees $c$ in transit, and like everybody else she knows $n$ and $e$. She would like to decrypt $c$. The most straightforward way to do it would be to find out what $d$ is.[63]

$d$ is a multiplicative inverse modulo $\phi(n)$ of $e$. So she would like to find out what $\phi(n)$ is. She knows that $\phi(n) = (p-1) \times (q-1)$, where $n = p \times q$. So she would like to factor $n$ into its two primes. How to prevent this: *Choose $p$ and $q$ to be large primes.* We do not know of a feasible way of factoring a product of two large primes to obtain these primes.

### $m$ is too big

If $m \geq n$, say $m = \alpha + \beta \times n$, where $0 \leq \alpha < n$ and $\beta \geq 1$, then both $m$ and $\alpha$ will decrypt to the same value as:

$$m^d \bmod n = \alpha^d \bmod n$$

[63]We are now discussing only the most straightforward approach

So what was the message: $m$, $\alpha$, or maybe something else. So we need to standardize: *always* $0 \leq m < n$.

### $m$ is too small

Assume that $m^e < n$ (not modulo anything, but actually). Then $c = m^e \bmod n = m^e$.

From here Eve can easily find $m$ by doing $m = c^{1/e}$, actually, not modulo anything. Therefore: *$m$ should be large (but of course smaller than $n$).*

### Same (encrypted) message is sent to some user repeatedly

Assume that Bob sends the same message to Alice repeatedly, for instance almost every day: "I am fine today." Then, if Alice behaves the same after each such message, Eve can predict her behavior based on the message, even if Eve cannot decrypt it.

*Do not send the same message twice to the same individual.*

### Users share $e$ and the same (encrypted) message is broadcast (sent) to many of them

It is actually common for everybody to have the same $e$. Originally, $e = 3$ (really!) was popular. Now it is considered insecure, so $e = 2^{16} + 1$ is used frequently.[64]

We will go over a relevant issue by means of a very small example. The problem occurs because of the Chinese Remainder Theorem. You can read about it in F[274–275], but it is not necessary to do so in order to understand what the problem is—I will give you the intuition behind it.

**Example 8.** This is an example of what Chinese Remainder Theorem is. Consider two relatively prime numbers: 3 and 4. Their product is 12. We consider all numbers 0, 1, . . . , 11, and create a table, see Fig. 40

Note that the pairs
$\langle 0 \bmod 3, 0 \bmod 4\rangle$
$\langle 1 \bmod 3, 1 \bmod 4\rangle$
. . .
$\langle 11 \bmod 3, 11 \bmod 4\rangle$
are all different.

[64]If written out it has only two non-zero bits, which makes some operations fast, as we will see later.

| $z$ | $z \bmod 3$ | $z \bmod 4$ |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 1 |
| 2 | 2 | 2 |
| 3 | 0 | 3 |
| 4 | 1 | 0 |
| 5 | 2 | 1 |
| 6 | 0 | 2 |
| 7 | 1 | 3 |
| 8 | 2 | 0 |
| 9 | 0 | 1 |
| 10 | 1 | 2 |
| 11 | 2 | 3 |

Figure 40: Residua modulo 3 and 4 of all integers smaller than 12

So given a pair $\langle x, y \rangle$, such that $0 \le x < 3$ and $0 \le y < 4$ there is a unique $z$, such that $0 \le z < 12$, and $x = z \bmod 3$ and $y = z \bmod 4$. In fact there is a fast algorithm to compute such a $z$, which we do not present.

□

Now, as we said it is common to use the same $e$ for everybody, or at least many people can share the same $e$. (Of course, they all have to have different $n$'s as otherwise they would share the same $d$.)

Assume, for this example that $e = 3$ and the same message $m$ was sent to 3 people, whose public keys are: $\langle n_1, 3 \rangle$, $\langle n_2, 3 \rangle$, and $\langle n_3, 3 \rangle$. Of course all of $n_i$'s will be relatively prime (with extremely high probability) as there were chosen randomly.

Then what was sent was: $m^3 \bmod n_1$, $m^3 \bmod n_2$, and $m^3 \bmod n_3$.

Since $m < n_i$, for each $i$, it follows that $m \times m \times m < n_1 \times n_2 \times n_3$.

Let $M = m \times m \times m$ and $N = n_1 \times n_2 \times n_3$.

Then we have exactly the setting for The Chinese Remainder Theorem, but for 3 moduli, not 2 as in our example above.

Indeed, $M < n_1 \times n_2 \times n_3$, and $c_1 = M \bmod n_1$, $c_2 = M \bmod n_2$, $c_3 = M \bmod n_3$.

And Eve can recover $M$. From this she gets $m = M^{1/3}$

Therefore, *never encrypt the same message more than once.*[65]

We will make it even stronger: *We would like to make sure (or help people to make sure) that no two messages no matter from whom and to whom sent*

[65] In our example two times may be OK, but not three times.

*are ever the same.*

### Signing "arbitrary messages"

When Alice signs, this means that she decrypts something with her private key, as she is the only one capable of doing this.

We have discussed in Sections 4.4–4.5 various issues that need to be addressed while digitally signing.

So she cannot just decrypt arbitrary strings presented to her

### Forging a signature on a related message

Here is another issue.

Mallory can forge Alice's "decryption" on a related string.[66] Assume that Alice as part of a signature procedure computes $D^{\text{RSA}}(d_{\text{A}}^{\text{RSA}}, x \bmod n)$ for some $x$. This is then seen by Mallory. Let's write $d$ for $d_{\text{A}}^{\text{RSA}}$

Note that

$D^{\text{RSA}}(d_{\text{A}}^{\text{RSA}}, x \bmod n) = x^d \bmod n$

But, also: $(x^2 \bmod n)^d \bmod n = (x^d \bmod n)^2 \bmod n$

Therefore, Mallory, who sees $x^d \bmod n$, can easily produce $(x^2)^d \bmod n$. In other words, given Alice's signature on $x$, he can easily forge Alice's signature on $x^2$ (or $x^2 \bmod n$).

So, of course Alice will "decrypt" the hash of the message and the above attack will not work. But we may also want to make sure in even stronger way that strings to be decrypted (as part of signing) are of a very specific format so the attacks above will not work. So we could say, that we want to decrypt not only a hash, but a hash of a very specific format.

## 7.4 A standard for encrypting and signing

Let us look now at a standard, that is not the newest because the new one is quite complicated and the one we look at provides some important intuition into how things are done in practice. For a more elaborate discussion both of the attacks and a standard, which we will not cover, see F[305–312]. The discussion there is confined to encryption only, but we will cover signing too.

The length of each record in Fig. 41 and Fig. 42 is the same as the length of $n$, which some integer number of octets.[67]

[66] Let us not assume for now that she decrypts a hash as part of producing a signature.

[67] Octet is a sequence of 8 bits. Commonly the term "byte" is used, but technically byte refers to a unit of access to the memory, which could be a different number of bits, though currently the standard is 8 bits. It is all very pedantic, but sometimes you will see "octets"

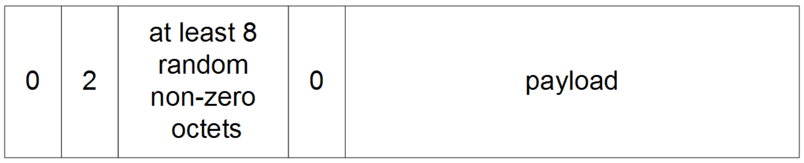


Figure 41: A format for the record to be encrypted.

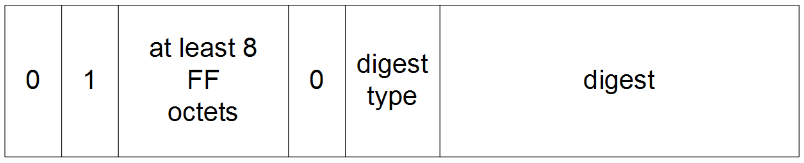


Figure 42: A format for the record to be decrypted to produce a signature.

Let us first discuss encryption.

The record in Fig. 41 stores some integer, say $x$. Let us examine the fields of the record, going from left to right:

1. Makes sure that $x < n$
2. Two purposes:
   (a) Indicates format/record type
   (b) Makes sure that $x$ is big
3. Makes sure that no two messages ever sent, no matter from whom to whom, are equal; non-zero so we do not get confused where the separator (see next) is in the record
4. Separator
5. Payload (generally a secret key for symmetric encryption, but could be anything.

Payload is the actual message we are interested in recovering. It is clear how to get it from the encrypted record $c$ by decrypting it and finding the payload field in it.

The record in Fig. 42 stores some integer, say $x$. Let us examine the fields of the record, going from left to right:

not “bytes” in formal specifications.

1. Makes sure that $x < n$
2. Two purposes:
   (a) Indicates format/record type
   (b) Makes sure that $x$ is big
3. Makes sure that $x$ is of very special format, so it is infeasible to forge signatures (as Mallory did in the example above, producing of forged signature on $x^2$).
4. Separator
5. Specification which function is used for the one-way-hash (there are various choices, as we will see: SHA-1, MD5, etc.).
6. One-way hash (of something to be signed)

## 7.5 Fast Exponentiation

**F[279–280]** We incorporate these. I will give a slightly different presentation of essentially the same algorithm, though expressed differently. (Note; F seems to be using $k$ and $n_\mathrm{b}$ to denote the same thing.)

The question is, how to compute $a^x$ fast. (We actually need results modulo some $n$, but we will return to this later, for now, think of standard integer arithmetic.)

If we write $x$ in binary it is some string of bits: $x_k x_{k-1} \ldots x_0$.

We will use the following notation: given some some string of bits $x_k x_{k-1} \ldots x_0$ (we will always assume that $x_k = 1$), the integer corresponding to it will be denoted by $\mathrm{num}(x_k x_{k-1} \ldots x_0)$, that is:

$$\mathrm{num}(x_k x_{k-1} \ldots x_0) = \sum_{i=k}^{0} x_i \times 2^i \tag{26}$$

From this, we make two claims (simple, so no proof needed):

1. $\mathrm{num}(x_k x_{k-1} \ldots x_0 0) = 2 \times \mathrm{num}(x_k x_{k-1} \ldots x_0)$
   Example: $\mathrm{num}(1010) = 2 \times \mathrm{num}(101)$
2. $\mathrm{num}(x_k x_{k-1} \ldots x_0 1) = 2 \times \mathrm{num}(x_k x_{k-1} \ldots x_0) + 1$
   Example: $\mathrm{num}(1011) = 2 \times \mathrm{num}(101) + 1$

So we conclude:

1. $a^{\text{num}(x_k x_{k-1} \ldots x_0 0)} = a^{2 \times \text{num}(x_k x_{k-1} \ldots x_0)} = \left(a^{\text{num}(x_k x_{k-1} \ldots x_0)}\right)^2$

2. $a^{\text{num}(x_k x_{k-1} \ldots x_0 1)} = a^{2 \times \text{num}(x_k x_{k-1} \ldots x_0)+1} = a \times \left(a^{\text{num}(x_k x_{k-1} \ldots x_0)}\right)^2$

We can use this for the algorithm to produce $a^c$ iteratively. Of course, if we want ultimately to get the result modulo some $n$, we should take this modulo after each operation so numbers do not grow unnecessarily. The algorithm is presented in Fig. 43.

```
procedure FastExponentiation($a, x$)                    ▷ $a^x$ mod $n$;    $x = \text{num}(x_k \ldots x_0)$
    $y \leftarrow 1$
    for $i \leftarrow k, k-1, \ldots, 0$ do
        $y \leftarrow y \times y$ mod $n$
        if $x_i = 1$ then
            $y \leftarrow a \times y$ mod $n$
        end if
    end for
end procedure
```

Figure 43: Algorithm for fast exponentiation

Let us run the algorithm on an example using the same numbers that F has in Example 9.7.

**Example 9.** $a = 17$, $c = 22$, and $n = 21$. First we note that: $22 = \text{num}(10110)$, so $k = 4$.

We will start accumulating the answer in $y$. Initially $y = 1$. We describe the progression of the computation in the table shown in Fig. 44, where we show how to compute $17^{22}$ mod 21.

□

Forouzan presents essentially the same algorithm, but "goes from left to right" in the exponent, where we "go from left to right." It is somewhat easier, in my opinion, to explain the algorithm and why it works in the form I present it—but the difference is not essential.

| | | Squaring | Multiplying |
|---|---|---|---|
| $i$ | $x_i$ | $y$ | $y$ |
| 4 | 1 | | 17 |
| 3 | 0 | $17^2 \bmod 21 = 16$ | 16 |
| 2 | 1 | $16^2 \bmod 21 = 4$ | $17 \times 4 \bmod 21 = 5$ |
| 1 | 1 | $5^2 \bmod 21 = 4$ | $17 \times 4 \bmod 21 = 5$ |
| 0 | 0 | $5^2 \bmod 21 = 4$ | 4 |

Figure 44: Computing $17^{22} \bmod 21$.

## 7.6 Primality testing

See Forouzan[261–266]. We will describe the Miller-Rabin algorithm. I will provide a different variant than what Forouzan shows,[68] because what I describe is easier to understand at the cost of a slightly reduced efficiency.

We want to look for properties that prime numbers always have, though these properties sometimes may be true for composite numbers too. We will want to decide whether some $n$ is a prime number. So we will apply many tests to $n$ to check if it has properties that all prime numbers have. If it passes all these tests, we will declare that with (sufficiently) high probability it is a prime number (because with very small probability $n$ is composite and still has these properties.

**Theorem 12.** If $n$ is prime then for every $a$, such that $1 \le a \le n-1$ we have $a^{n-1} \equiv 1 \bmod n$.

□

*Proof.* If $n$ is prime, then $\phi(n) = n-1$, and therefore this follows from Theorem 10.

□

**Conclusion** If for some $a$ and $n$, such that $1 \le a \le n-1$ we have $a^{n-1} \not\equiv 1 \bmod n$, then $n$ is not prime. We can use this to “disprove” primality. But note, that even for composite $n$, this equation could hold. For instance: $4^{15-1} \equiv 1 \bmod 15$, even though 15 is not a prime.

**Theorem 13.** If $n > 2$ is a prime, then the equation $x^2 \equiv 1 \bmod n$ has exactly two solutions in the range $[0, \ldots, n-1]$ and they are 1 and $n-1$. (Note that $n-1 \bmod n$ is the same as $-1 \bmod n$, so we could have said that there are only two solutions $-1$ and $+1$.)

[68]He presents the more common one

Another way of saying this is to say that if $n$ is a prime then 1 has exactly two square roots modulo $n$. For composite $n$, 1 may have more than two square roots modulo $n$.

$\square$

Before we proceed to the proof, let us look at an example

**Example 10.** As Forouzan shows:

1. $n = 7$, 1 has two square roots: 1, 6 modulo $n$.
2. If $n = 8$, 1 has four square roots: 1, 3, 5, 7 modulo $n$.
3. If $n = 22$, 1 has two square roots: 1, 21 modulo $n$. (Only two, even though 22 is not a prime.)

*Proof.* Let $n > 2$ be a prime.

We first show that 1 and $n - 1$ are square roots of 1 modulo $n$.

1. 1 is a square root because:

   $1^2 \bmod n = 1 \bmod n = 1$
2. $n - 1$ is a square root because:

   $(n-1)^2 \bmod n = n^2 - 2 \times n + 1 \bmod n = (n-2) \times n + 1 \bmod n = \alpha \times n + 1 \bmod n = 1$

We now show that there are no other square roots of 1 modulo $n$. We are looking for $x$'s such that

(27) $x^2 = 1 + \alpha \times n$

This is of course the same as

(28) $(x - 1) \times (x + 1) = \alpha \times n$

Let $x_0$ satisfy Eq. 28. Then of course

$$(x_0 - 1) \times (x_0 + 1) = \alpha \times n$$

Therefore, $(x_0 - 1)$ and/or $(x_0 + 1)$ are divisible by $n$.

We will show that both of them cannot be divisible by $n$. Assume otherwise by contradiction. Then we have

$$\begin{aligned} x_0 - 1 &= \beta \times n \\ x_0 + 1 &= \gamma \times n \end{aligned}$$

Of course, $\gamma > \beta$. We can write:

$$\begin{aligned} (x_0 + 1) - (x_0 - 1) &= (\gamma - \beta) \times n \\ 2 &= (\gamma - \beta) \times n \end{aligned}$$

But as, $\gamma - \beta \geq 1$ and $n > 2$, this is impossible. Therefore exactly one of $(x_0 - 1)$ and $(x_0 + 1)$ is divisible by $n$.

Let $x_1$ be a solution such that $x_1 - 1$ is divisible by $n$. Then $x_1 - 1 = \delta \times n$ and $x_1 = 1 + \delta \times n$.

Let $x_2$ be a solution such that $x_2 + 1$ is divisible by $n$. Then $x_2 + 1 = \delta \times n$ for $\delta \geq 1$ and $x_2 = -1 + \delta \times n$. Then we can also write $x_2 = n - 1 + (\delta - 1) \times n$, and $\delta - 1 \geq 0$. So we have shown that $x_2 = n - 1 + \epsilon \times n$

So we have shown that the only square roots of 1 modulo $n$ in the range $0, 1, \ldots, n - 1$, are indeed $+1$ and $n - 1$.

$\square$

**Conclusion** This can be used to test if $n$ is prime. Indeed, if $1 \leq a \leq n - 1$ satisfies $a \times a \bmod n = 1$, then $n$ cannot be a prime.

We will use the above observations to try and find out if some $n$ is a prime. We present essentially the Rabin-Miller probabilistic algorithm for primality testing:

1. If $n$ is prime, all the tests for primality will be passed
2. If $n$ is not prime, with very high probability (as high as we want, we just need to run the algorithm long enough) some test for primality will be failed.[69]

We will pick a random $a$, such that $0 < a < n$ and compute $a^{n-1} \bmod n$ using Fast Exponentiation. If $a^{n-1} \bmod n \neq 1$, then $n$ is not a prime. But

[69]Numbers that pass various tests for primality even though they are not prime, are called *pseudoprimes*. See also http://en.wikipedia.org/wiki/Pseudoprime. We do not need to worry about them in practice.

note that during the exponentiation we compute various squares, the values $y \times y \bmod n$ for various $y$'s. If for one of them we have that $y \times y \bmod n = 1$, but $y \neq 1$ and $y \neq (n-1)$, then we also can determine that $n$ is not prime.

We could still be unlucky and none of the above happens and $n$ is still not a prime (is composite).

The algorithm is given in Fig. 45.

```
procedure PrimalityTesting($a, x$)                ▷ Computing $a^{n-1} \bmod n$
    $y \leftarrow 1$
    for $i \leftarrow k, k-1, \ldots, 0$ do
        $z \leftarrow y$
        $y \leftarrow y \times y \bmod n$
        if $y = 1 \wedge z \neq 1 \wedge z \neq (n-1)$ then
            $n$ is not prime                        ▷ bad square root
        end if
        if $x_i = 1$ then
            $y \leftarrow y \times a \bmod n$
        end if
    end for
    if $y \neq 1$ then
        $n$ is not prime                            ▷ bad final value
    else
        $n$ is "perhaps prime"
    end if
end procedure
```

Figure 45: Miller-Rabin algorithm for one value of $a$

How many times do we need to run the procedure to be reasonably sure that $n$ is a prime, getting each time the answer "perhaps prime"?

There is a theorem, which takes quite a while to prove, and therefore we will state it without a proof.

**Theorem 14.** Let $n$ be an odd integer and let a set of $m$ $a$'s, such that $1 < a < n$, and for each of these $a$'s the algorithm returns "perhaps prime." Then, $n$ is a prime number with probability of at least $1 - \frac{1}{2^m}$.

□

So if we want to be sure with probability of at least $1 - \frac{1}{2^{100}}$ that $n$ is prime, it is enough to get "perhaps prime" for 100 random $a$'s.

Let us now discuss how to find a large prime, say one of 2048 bits. We will create a string of 2048 bits, the first and the last being 1, and the "middle"

2046 chosen randomly. (You may as well choose each bit independently randomly with probability of 0.5 of it being 1.) Run Miller-Rabin on it with some number number of $a$'s. If all the answers are "perhaps prime," we are happy. Otherwise we try another random $n$, chosen as above.

We now need to ask how many primes there are. Another theorem we are not going to prove.

**Theorem 15.** (See also `http://en.wikipedia.org/wiki/Prime_number_theorem#Statement_of_the_theorem`.) The probability that a random $n$ is a prime is: $\approx \frac{1}{\ln n}$.

□

So if $n$ has 2048 bits (with the first bit being 1) then the probability that it is a prime is:

$$\approx \frac{1}{\ln 2^{2048}} = \frac{1}{2048 \times \ln 2} \approx \frac{1}{2048 \times 0.7} \approx 0.0007$$

From this we can conclude that to find a number of 2048 bits so that we are sure with probability of at least $1 - \frac{1}{2^{100}}$ that it is a prime will take on the average about

$$100 \times \frac{1}{0.0007} \approx 71,000$$

runs of the Miller-Rabin test.

It is all completely feasible using current PCs, assuming of course that you get good random numbers.[70]

Figures 46 and 47 show executions of our version of the algorithm on two examples from Forouzan.

| $i$ | $x_i$ | $z$ | $y$ | $y$ |
|---|---|---|---|---|
| 4 | 1 | 1 | 1 | 2 |
| 3 | 1 | 2 | 4 | 8 |
| 2 | 0 | 8 | 10 | 10 |
| 1 | 1 | 10 | 19 | 11 |
| 0 | 0 | 11 | 13 | 13 |

Figure 46: $n = 27$ and $a = 2$. Miller-Rabin says: 27 is not a prime, because $2^{26} \bmod 27 \neq 1$.

[70] We are not accounting for the effort to find such random bits.

| $i$ | $x_i$ | $z$ | $y$ | $y$ |
|---|---|---|---|---|
| 5 | 1 | 1 | 1 | 2 |
| 4 | 1 | 2 | 4 | 8 |
| 3 | 1 | 8 | 3 | 6 |
| 2 | 1 | 6 | 36 | 11 |
| 1 | 0 | 11 | 60 | 60 |
| 0 | 0 | 60 | 1 | 1 |

Figure 47: $n = 61$ and $a = 2$. Miller-Rabin says: 61 is perhaps prime. (It actually is a prime.)

## 7.7 Key ideas

1. $\phi(n)$ and its values when $n$ is prime and when $n$ is a product of two primes
2. $a^{k \times \phi(n)+1} \equiv a \bmod n$
3. RSA algorithm
4. RSA protocol
5. Two conditions that prime $n$ obeys
   (a) For any $a$, such that $0 < a < n$, we have $a^{n-1} \equiv 1 \bmod n$
   (b) The only square roots of 1 modulo $n$ are: 1 and $n - 1$ (the latter is really $-1$).
6. Miller Rabin algorithm for primality testing and its usage for finding large primes

## 8 Components for symmetric encryption (using block ciphers)

I very much like Forouzan presentation. It just has too much material for what we can and need to cover to understand the ideas. So I will just use the textbook, making only a small number of changes to the book.

**F[124]** (ignore example: we assume that the message has *exactly* $n$ bits, whatever the machine can encrypt "directly"). So for DES it is exactly 64 bits, as we will see later.

We now discuss the full/partial size keys (we are not taking this from Forouzan).

| message | cipher |
|---|---|
| 00 | 01 |
| 01 | 10 |
| 10 | 00 |
| 11 | 11 |

Figure 48: Messages and corresponding ciphers.

| string | label |
|---|---|
| 00 | a |
| 01 | b |
| 10 | c |
| 11 | d |

Figure 49: Labels for 2-bit strings

| message | cipher |
|---|---|
| a | b |
| b | c |
| c | a |
| d | d |

Figure 50: Labels of messages and of corresponding ciphers.

Alice and Bob want to exchange arbitrary 2-bit strings.

Let us consider encryption of 2 bit messages to get 2 bit ciphers. We need to specify a cipher for each possible message. An example of such a specification (by explicit writing it out) appears in Figure 48.

Note that two different messages have to have two different ciphers, as otherwise there is not a unique decryption.

Let us now label (or name) explicitly each 2 bit spring, as seen in Figure 49. Then, we can redo our example, as seen in Figure 50.

| | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| a | a | a | a | a | a | b | b | b | b | b | b | c | c | c | c | c | c | d | d | d | d | d | d |
| b | b | c | c | d | d | a | a | c | c | d | d | a | a | b | b | d | d | a | a | b | b | c | c |
| c | d | b | d | b | c | c | d | a | d | a | c | b | d | a | d | a | b | b | c | a | d | a | b |
| d | c | d | b | c | b | d | c | d | a | c | a | d | b | d | a | b | a | c | b | d | a | b | a |

Figure 51: Permutations of 4 elements (listed in columns).

When Alice and Bob communicate using an encryption, they in effect choose a table such as in Figure 50. Note that this is really a permutation of $\langle a, b, c, d \rangle$. How many such tables exist? As many as there are permutations. They are all effectively listed in Figure 51. Each column corresponds to a table. Our example permutation is in the 9th column.

The number of such permutations is $4! = 4 \times 3 \times 2 \times 1$,[71] since there are 4 choices of where $a$ will go, then 3 choices of where $b$ will go (one slot was taken by $a$), etc.

So how many bits do we need to specify such a permutation? We need to distinguish among 24 choices, so the number of bits is $\lceil \log_2 24 \rceil = 5$. If Alice and Bob are smart, they do not want in any way to "biase" the selection of the key, so they will choose it randomly, thus making it more difficult for Eve or Mallory to guess it.

Thus, as any one out of 24 keys is likely, we can say that Alice and Bob need 5 bits to specify a permutation and therefore they use a key of 5 bits.[72]

In general if the universe of messages has $k$ elements (labels) to permute, then there are $k!$ permutations.

If we want to encrypt messages of 64 bits, then there are $2^{64}$ possible messages, which correspond to the labels above. Therefore there are $2^{64}!$ permutations. And to specify which one we are going to use will take $\lceil \log_2(2^{64}!) \rceil$ bits. This will be the lengths of *full-size keys*, which are needed to be able to specify every possible permutation.

Let us estimate the various numbers appearing above.[73]

[71]Of course, "!" denotes factorial.

[72]Note that they need a rather "long" key of 5 bits to encrypt/decrypt rather "short" messages of 2 bits.

[73]In general, we will use crude estimates if they good enough to make our point, instead of looking up better estimates in various off-line or on-line handbooks. This is better for developing intuition.

$$\begin{aligned}
2^{64}! &= \overbrace{2^{64} \times (2^{64}-1) \times \cdots \times (2^{32}+1)}^{2^{32} \quad \text{terms}} \times 2^{32} \times (2^{32}-1) \times \cdots \times 1 \\
&> \overbrace{2^{32} \times 2^{32} \times \cdots \times 2^{32}}^{2^{32} \quad \text{terms}} \\
&= (2^{32})^{2^{32}} \\
&= 2^{32 \times 2^{32}}
\end{aligned}$$

So the number of bits needed to specify a key is $> \log_2 2^{32\times 2^{32}} = 32 \times 2^{32}$, which of course is impractically large.[74]

Therefore, any encryption method (machine) can only use a very small subset of the possible permutations, essentially $2^{\text{length of the key}}$.

For our example, let us say that Alice and Bob decide to use 1-bit keys. Here is what this means. There is a machine, completely known to everybody, which kind of looks like Fig. 50. What they do to use it, they agree in secret on one of the two keys, $k = 0$ or $k = 1$. Notice, that the only secret thing is the key! So Eve and Mallory know the two candidate permutations, just do not know which one of them is used by Alice and Bob.

| $k = 0$ | $k = 1$ |
|---|---|
| b | c |
| c | a |
| a | b |
| d | d |

Figure 52: Two permutations when key length is 1. One of two keys is chosen and specified using 1 bit.

In Fig. 53, we see the ciphers that the machine produces depending on the key fed to it.

DES uses 56-bit keys,[75] and therefore can only specify $2^{56}$ permutations. Therefore, DES uses *partial-size keys*.

Back to the book:

**F**[**128,**−7]–**F**[**132,15**]

**F**[**142–143**] Final design of Feistel cipher: encrypting and decrypting.

[74]Actually, the key must be much larger than what we estimated using our very simple computation, but we do not need to know what it is to make our point that it is not practical.

[75]Not 64 bits, as we will see.

| message | cipher when $k = 0$ | cipher when $k = 1$ |
|---|---|---|
| 00 | 01 | 10 |
| 01 | 10 | 00 |
| 10 | 00 | 01 |
| 11 | 11 | 11 |

Figure 53: Encryption of messages under the two possible encryption keys.

## 8.1 How does Feistel cipher work

It is in Forouzan, pages 142–143, but am presenting it here too. Look at Fig. 54, which is really identical (with different notation) to Forouzan's Figure 5.17.

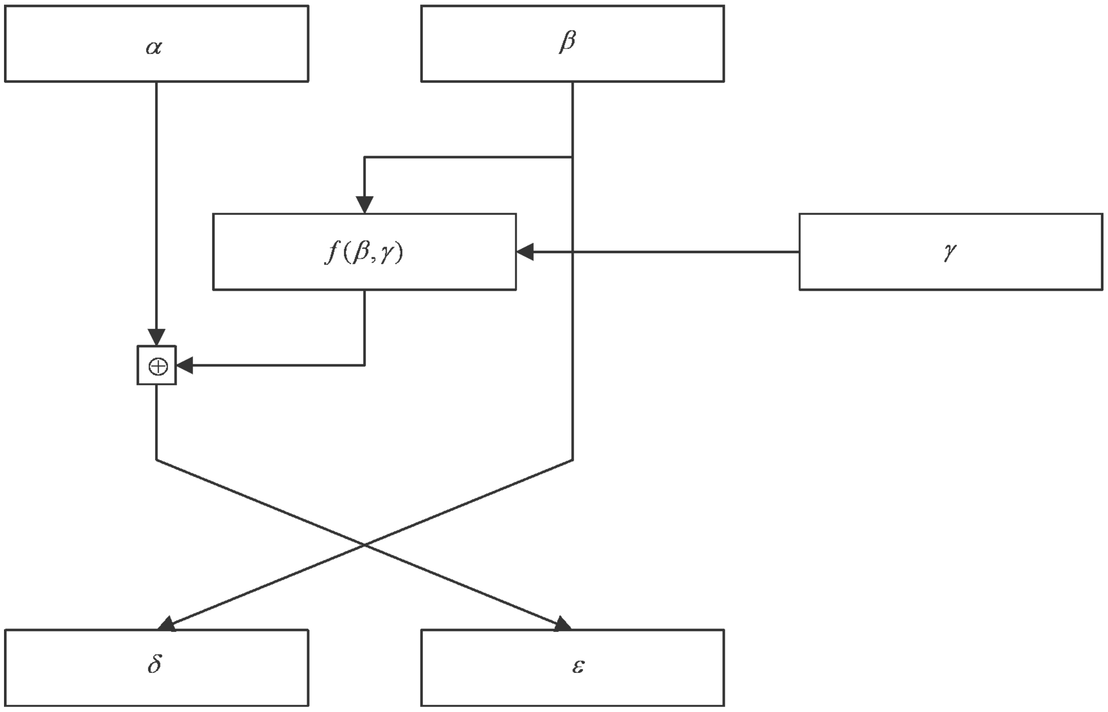


Figure 54: A part of Feistel's cipher.

Alice and Bob share a secret key (and this is the only thing say share). Alice encrypts a message, and sends the cipher (which everybody can see) to Bob. Bob wants to decrypt it, to get the original message. We will see how Bob "walks back" through one stage.

Let us see who knows what:

1. Structure of the machine, including $f$, is known to everybody
2. $\alpha$ and $\beta$ are known only to Alice
3. $\delta$ and $\epsilon$ are known to all
4. $\gamma$ is derived in a known way to all from the secret key known only to Alice and Bob

Here is what Bob does to commpute $\alpha$ and $\beta$:

1. $\beta$ is just $\delta$, and therefore known
2. As $\gamma$ is known (this is obtained from the secret shared key in a completely deterministic manner—manner that is publicly known), he computes $f(\beta, \gamma)$.
3. Now, we he can compute $\alpha$:

$$\begin{aligned}\epsilon \oplus f(\beta,\gamma) &= \big(\alpha \oplus f(\beta,\gamma)\big) \oplus f(\beta,\gamma)\\ &= \alpha \oplus \big(f(\beta,\gamma) \oplus f(\beta,\gamma)\big)\\ &= \alpha\end{aligned}$$

## 8.2 Key ideas

1. Fundamentals of symmetric encryption
2. Partial size keys vs. full size keys
3. P-boxes
4. S-boxes
5. Feistel cipher

# 9 DES: Data Encryption Standard

**F[159–175]** Just get the flavor of the specifications and ignore formal listings of the algorithms. I will cover the key ideas of how this works in class.

**F[175–176]** Properties (I will discuss briefly)

**F[178]** Key size

**F[182,**−6]–**[185,Figure 6.16]** I will discuss the "meet-in-the-middle" attack in much more detail, see material below.

## 9.1 Meet-in-the-middle-attack on 2DES

**Example 11.** We consider 2DES for keys of length 2 and messages of length 3. The same key is used for encryption and decryption. For convenience, we will refer to the keys by labels: 1, 2, 3, 4 (technically in bits they are: 00, 01, 10, 11.). For convenience, we will refer to messages by labels 1, . . . , 8 (technically in bits they are: 000, . . . , 111.).

Therefore, there are 16 choices for the pair of two keys used, ranging from $\langle 1, 1\rangle$ to $\langle 4, 4\rangle$.

We are given two pairs of the form $\langle \text{message}, \text{cipher}\rangle$: $\langle m_1, c_1\rangle$, $\langle m_2, c_2\rangle$.

We will start our "meet in the middle" attack. We take $m_1$ and apply all possible keys to it getting 4 encryptions. This is depicted in Figure 55.[76]

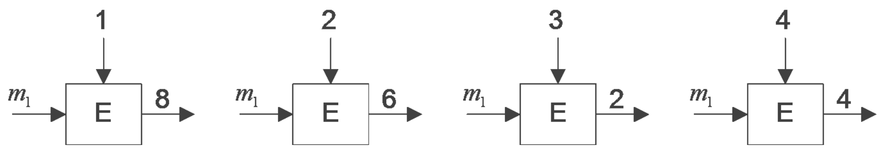


Figure 55: Encrypting $m_1$ with all possible encryption keys.

We take $c_1$ and apply all possible keys to it getting 4 decryptions. This is depicted in Figure 56. (We do not run anything in reverse, it is just intuitively convenient to think of the input to the decrypting machine entering from the right.)

Another way of describing what we have is in Figure 57.

We now sort columns 1 and 2 of this figure by column 2 and we sort columns 3 and 4 by column 3, and we get Figure 58.

This allows efficient matching. Note that encryption of $m_1$with 3 produces the same string as the decryption of $c_1$ with 1 and encryption with 1

[76]Of course, for the example, we invent the results.

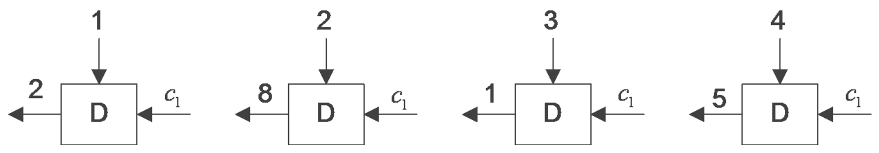


Figure 56: Decrypting $c_1$ with all possible decryption keys.

| encryption key | encrypted message | decrypted cipher | decryption key |
|---|---|---|---|
| 1 | 8 | 2 | 1 |
| 2 | 6 | 8 | 2 |
| 3 | 2 | 1 | 3 |
| 4 | 4 | 5 | 4 |

Figure 57: Encrypting $m_1$ and decrypting $c_1$

| encryption key | encrypted message | decrypted cipher | decryption key |
|---|---|---|---|
| 3 | 2 | 1 | 3 |
| 4 | 4 | 2 | 1 |
| 2 | 6 | 5 | 4 |
| 1 | 8 | 8 | 2 |

Figure 58: Encrypting $m_1$ and decrypting $c_1$, with the table rearranged

produces the same string as decryption with 2. This implies what is shown in Fig. 59.

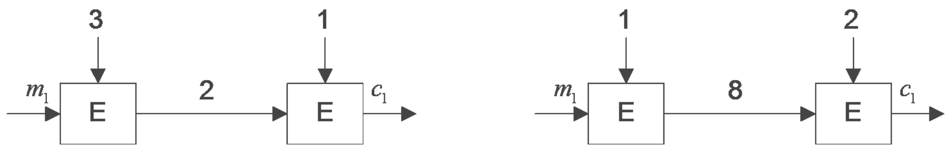


Figure 59: Two key pairs consistent with the pair $\langle m_1, c_1 \rangle$.

So we conclude that the only possible key pairs *used for 2DES encryption* could be $\langle 3, 1 \rangle$ and $\langle 1, 2 \rangle$. We need to find which one of them is the one Alice and Bob use. We have another pair we can use: $\langle m_2, c_2 \rangle$. We could use another "meet in the middle" attack, but we only have 2 possible key pairs, so let us just test both of them by using them to encrypt $m_2$ and checking whether we get $c_2$ or not. The results are in Fig. 60.

So we conclude that the only possible key pair was $\langle 3, 1 \rangle$, as it was the only one consistent with both the given $\langle m_1, c_1 \rangle$ and $\langle m_2, c_2 \rangle$.

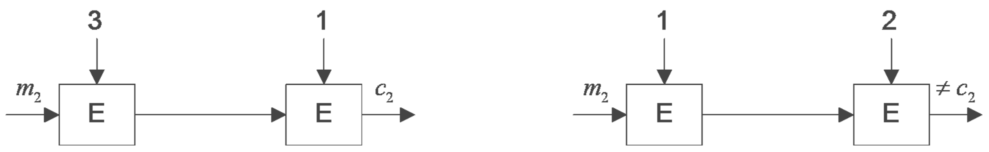


Figure 60: Applying the two remaining key pairs to $m_2$ (the "intermediate" values between the two encryptions are not important).

$\square$

Let us now analyze the efficiency of this attack. There will be two stages to the attack in the general case just like in the example above (with extremely high probability, we will need exactly two stages.) We want to find the key pair used $\langle k_1, k_2 \rangle$.

In the first stage we look at all the $2^{56} \times 2^{56} = 2^{112}$ key pairs, over all encryption keys and all decryption keys. So we ask how many pairs of the form $\langle e, d \rangle$ we have such that:[77]

$$E(e, m_1) = D(d, c_1)$$

Pick a *specific* pair $\langle e, d \rangle$. This produces a specific $E(e, m_1)$ a specific string out of $2^{64}$ possible. Consider the specific $D(d, c_1)$. This is also a specific string out $2^{64}$ possible. The probability that the two strings are equal is $\frac{1}{2^{64}}$. Why? Because after getting $E(e, m_1)$, a specific 1 out of $2^{64}$ strings, we take some specific $D(d, c_1)$ and as it is also a specific string out of $2^{64}$ strings, the probability that it happens to be $E(e, m_1)$ is only $\frac{1}{2^{64}}$.

We can look at this as the expected number of "collisions" that a particular pair generates—the expected number is $\frac{1}{2^{64}}$.

But, we have a total of $2^{112}$ key pairs, therefore the expected number of collisions is:[78]

$$2^{112} \times \frac{1}{2^{64}} = 2^{48}$$

These are the "random" collision. But there is one pair, which is not random, this is the actual key pair used for 2DES application to $m_1$ to get $c_1$

We find them by sorting as described in F and in our example and matching on the "meet in the middle" value.

[77]This requires stating a little more carefully what exactly is our probabilistic model, which we do not do here.

[78]This is the number of key pairs multiplied by the probability that a pair produces a "collision."

We now apply the the $2^{48}$ key pairs to the the second pair of $\langle m_2, c_2 \rangle$.

The expected number of collisions (if everything is random) is by the same argument as above:

$$2^{48} \times \frac{1}{2^{64}} = \frac{1}{2^{16}}$$

In other words, there are no random collisions left (with extremely high probability). But there is one non-random one, the actual key pair $\langle k_1, k_2 \rangle$.[79] And this will be the only one left.[80]

## 9.2 Triple DES

**F[184-185]**

## 9.3 Key ideas

**F[184, Summary]**

[79]Recall that an encryption key and a decryption key are the same

[80]If there are more than two "collisions" left, we need another pair $\langle m_3, c_3 \rangle$.

# 10 Advanced Encryption Standard

We will look at a “picture-like” description, taken from **F[Chapter 7]**. It is actually enough (in my opinion) to study some figures and tables. Specifically, we will look at discuss

1. Figures: 7.1, 7.3, 7.4, 7.5, 7.6, 7.7, 7.8, 7.9, 7.10, 7.13, 7.15, 7.16
2. Tables: 7.1, 7.2

## 10.1 Key ideas

1. Longer keys, more secure than DES
2. Designed to thwart some attacks (we did not discuss specifics)

# 11 Encryption using symmetric-key systems

This is important, but we can cover all we need to know by analyzing some figures from the textbook and by looking at a few paragraphs. The main question discussed here is how to encrypt messages that are not exactly one “block,” such as 64 bits for DES. These could be long messages, or messages that trickle in, say one byte at a time. Various issues will be discussed.

Figures and paragraphs will refer to the textbook.

## 11.1 ECB

Figure 8.2, Security Issues, Error Propagation, Applications

## 11.2 CBC

Figure 8.3, IV, Security Issues, Error Propagation, Applications

## 11.3 CFB

Note, this is one-time pad generated pseudorandomly (even if K and IV are random) as they are seeding a deterministic process for generating a long key.

Figures 8.4, 8.5, Security Issues, Error Propagation, Applications

## 11.4 OFB

Figures 8.6, 8.7, Security Issues, Error Propagation

## 11.5 CTR

Figures 8.8, 8.9, Security Issues, Error Propagation

Table: 8.1

## 11.6 RC4

Figure 8.10 (just glancing at it) and Algorithm 8.6

## 11.7 Key ideas

1. Various block and stream encrypting modes: advantages and disadvantages as listed

## 12 PGP and secure mail

PGP is a system for sending secure mail. There is an open source free version, which you should download, install, and experiment with. I will provide the specific assignment in a separate document.

People know other people's public keys, and their own private (and public) keys. We will go over this briefly later relying on Forouzan. However, I will provide the key ideas here separately.

Alice wants to send a (long) email message to Bob. She may want to encrypt and/or sign the message. We know how it is generally done, from Section 4.7.

Let us be more specific. Alice and Bob share the following:[81]

1. Public/private cryptosystem, say RSA
2. Symmetric cryptosystems, say AES
3. A one-way-hash, say SHA-512,[82] we will denote it by $h$
4. Some lossless compression procedure, say ZIP, we will denote it by $Z$
5. A procedure for converting the message to ASCII,[83] producing also lines of length 64, we will denote it by $R$

Alice wants to send $x$ to Bob. $e$ will refer to Bob's public key and $d$ will refer to Alice's private key. The procedure that PGP employs for creating the message to be sent, denoted by $y$, is described in Fig. 61. The symbol $||$ denotes concatenation of strings.

An interesting point, how does Alice store $d$? It is not stored as it is, but it is encrypted. She has a passphrase, which is a long string that is difficult to guess but she can easily remember, such as "When I was a child I liked eating broccoli but hated apples". Call this phrase: $f$. Then her $d$ is stored as $g = E^{\text{AES}}(h(f), d)$. That is the system takes $f$ and produces $h(f)$. This $h(f)$ is used as a symmetric key to encode her private key $d$.[84] So if Eve looks at her disk, where Alice stores her private key, Eve cannot determine

[81]Generally they may share a *suite* of protocols, such being able to use both 3DES and AES for symmetric encryption. The sender chooses the ones to use as there is no previous negotiation between the participants.

[82]More about it later.

[83]Mailers like messages of ASCII characters (in lines that are not too long). Otherwise MIME is used, generally.

[84]We assume that the number of bits in $h(f)$ is the length for the symmetric key; this can always be arranged by "cutting down," etc. Also, note that any string of bits can

```
1: procedure AliceSendsAMessageToBob($x$)
2:     $y \leftarrow x$
3:     if Signature Needed then
4:         $y \leftarrow y||D^{\mathrm{RSA}}(d, h(y))$
5:     end if
6:     $y \leftarrow Z(y)$
7:     if Confidentiality Needed then
8:         Generate Random AES Key: $k$
9:         $y \leftarrow E^{\mathrm{RSA}}(e, k)||E^{\mathrm{AES}}(k, y)$
10:    end if
11:    $y \leftarrow R(y)$
12: end procedure
```

Figure 61: Alice wants to send to send $x$ to Bob using PGP. She actually sends $y$ to Bob.

$d$ because it is encrypted by an unknown (to Eve) symmetric key. This is how PGP generally finds out what the private key is for signing by Alice (with Alice's private key) and decrypting by Bob (with Bob's private key), see Fig. 62.[85]

```
1: procedure PGPRetrievesPrivateKey($f, g$)
2:     $t \leftarrow h(f)$
3:     $d \leftarrow D^{\mathrm{AES}}(t, g)$
4: end procedure
```

Figure 62: PGP needs the private key $d$. Actually, its encrypted version $g$ is stored. Alice keys in her passphrase $f$ and its hash $h(f)$ is used by PGP as a symmetric key for decrypting $g$ to obtain $d$.

## 12.1 Keyrings and trust model

Very briefly, each user has a public key ring of public keys of the various users and a private keyring of various private keys. (For instance my wife and I

serve as a key, so $h(f)$ can be a key; parity as in DES can be easily fixed by cutting $h(f)$ to 56 bits and expanding it to 64 bits by taking the correct parity bits. DES was just an example for discussion here.

[85]Of course, $g$ was originally obtained once $d$ and $f$ were chosen. Note that $f$ can be changed later by Alice, without $d$ being changed. The system simply replaces $g$ by encrypting $d$ with the new symmetric key.

have both have our private keys. So she can use my computer and I can use hers. But I cannot use her private key as I do not know her passphrase.

There is no central certificate authority, so there are various ways in which I need to figure out how I can trust that Alice's public key is indeed Alice's. You can read about it in F[Chapter 16], though I do not think it is necessary in practice (at least for me). I communicate confidentially with only a small number of people, whom I know and can trust their public key.

## 12.2 Key ideas

1. How PGP works

# 13 Cryptographic hash functions

See, Forouzan[Chapter 12]. We want to get a one-way hash, where strings/messages $M$ of arbitrary length are input into a completely specified machine $H$ and strings of fixed length, say 160 or 512 bits are produced, see Fig. 63. The desired properties were previously described in Section 4.2.

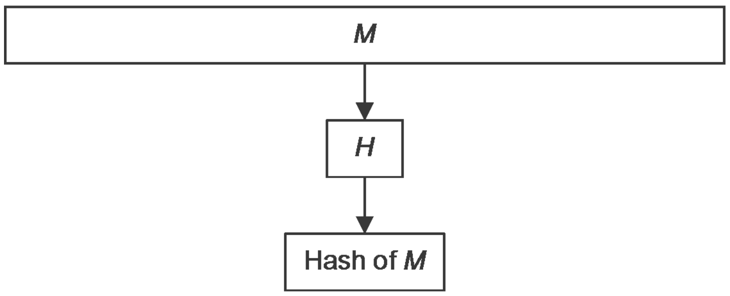


Figure 63: Hashing a long message, very schematic description.

In this chapter various such functions are described. Relying on Forouzan, and looking at some of the figures in the book, particularly 12.1, 12.6, 12.8–12.11, 12.15, and the associated tables we will get a sufficient feeling on what they look like. We do not need to understand the details of these constructions.

It is more important to understand how they are used, which we will do next.

## 13.1 Key ideas

1. Basic ideas underlying some standard cryptographic hash functions
2. Sketches of SHA-512 and Whirlpool

# 14 Message integrity and message authentication

The material appears in F[Chapter 11] with some proofs in Appendix E. I will extract and present here what we need to know, so looking at F is optional.

## 14.1 Obtaining digests from one-way hash functions

The basic idea is as follows. Alice wants to send a message $M$ to Bob. She also wants Bob to be sure that the message has not been corrupted and/or was really sent by her.

Alice and Bob share some one-way hash function $H$. Alice does the following:

1. Alice produces $H(M)$. In our context, we may refer to this hash as "digest" or "fingerprint" of $M$
2. Alice sends the message and its digest to Bob

Bob does the following

1. He computes the hash of the message
2. He compares the hash to the digest. If they are equal, he is satisfied that the message he received was "authentic."

Message integrity and message authentication are quite related. Basically:

1. Message integrity means that the message is not corrupted, assuming that the digest is not corrupted
2. Message authentication means the above and also that we can be sure who created the message

To better explain the concepts and the ideas, we will repeat some of what we have done before.

We want to create a fixed size (say 160 bits long) message digest. We will use a one-way hash function to do it.

We have a "strange" function $f$, which takes a block of some number of bits, say $m$ and produces $m$ bits. That is: $f\colon 2^m \to 2^m$.[86]

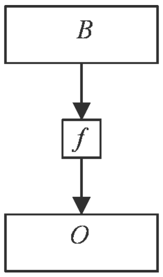


Figure 64: $f$ produces a digest of block $B$ by hashing it and producing $O$.

This $f$ will be a one-way hash. We can look at Fig. 64. Nothing particularly interesting in this figure, but it will be a building block for the rest.

Similarly, to what we did when encrypting long messages, we need to discuss how to hash/digest long messages. Our first attempt is to use chaining, as shown in Fig. 65. Here, the message consists of $n$ blocks of $m$ bits each, denoted by $B_1, B_2, \ldots, B_n$.[87]. So the message is $M = B_1 B_2 \ldots B_n$. The digest of the entire message, $h(M)$ is actually $O_n$.

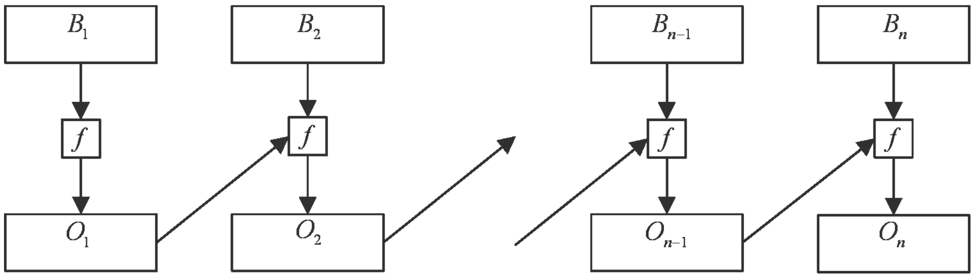


Figure 65: Digesting a long message, the digest is $O_n$.

Note the similarity to encrypting long messages. The differences are:

1. There is no secret key (but there may be one later, as we will see)

2. We are only interested in the last block obtained and throw out the rest

[86]Actually this could be $f\colon 2^{m_1} \to 2^{m_2}$, with $m_1$ and $m_2$ constant, but not necessarily equal, for instance $m_1 = 1024$ and $m_2 = 512$ but this is not really important.
[87]Padding is added to get an integral number of blocks.

An alternative way of looking at the same process, when we are not interested in talking about the internal structure (or perhaps the internal structure is in fact different) would be as depicted in Fig. 63, which we have seen before, but we may to use the term "digest" of "fingerprint," instead of "hash."

So Alice sends the hash in some secure way to Bob, and when Bob receives both $M$ and $g = H(M)$, he can confirm that he got $M$ without any errors by computing $H(M)$ and confirming that it is $g$. See also, the example we had earlier, in Section 4.3.1: `http://www.gimp.org/downloads/`. The "MD5 sum" is a cryptographic hash function, see `http://en.wikipedia.org/wiki/MD5`.

Of course, we need to be sure that $g$ is not corrupted and that Mallory did not substitute his own $M'$ and $g' = H(M')$ for the original $M$ and $g = H(M)$.

We can think of two ways of preventing Mallory from doing this:

1. If Alice has a public key and it is known to Bob, Alice can just decrypt the $g$ and send this decryption instead of $g$. This is how we have signed messages previously

2. $g$ can be sent to Bob in a different secure way, such as perhaps telling him what it is on the phone

But later, we will see additional ways of making sure that the digest is not corrupted, relying on symmetric keys.

## 14.2 The "birthday paradox"

We will discuss here some implications of the need to have strong collision resistance.

We need to discuss how easily can Mallory produce *another message $M'$ with the same hash as $M$*, that is: $H(M) = H(M')$. If he can produce such an $M'$, Mallory can convince Bob that $M'$ is the original message $M$, as its hash is the same as that of $M$. Thus convincing Bob to accept a forged message. This is best analyzed by means of a so-called *birthday paradox*.[88].

Forouzan has a much more extensive discussion of the various types of the birthday paradox, which are are perhaps worth reading, see F[345–352] and F[611–614]. But for our purpose it will be enough to build the intuition with a simpler (but not quite right mathematically) discussion.

[88]It is really not a paradox at all

Consider $k$ boys and $k$ girls. What is the probability that there is a pair $\langle$boy,girl$\rangle$, that share a birthday?[89] Sharing a birthday means that they were born in the same month and on the same day, but not necessarily in the same year. We will assume a year of 365 days.

We actually want to ask "how big should $k$ be so that the probability of at least one such pair sharing a birthday is at least 0.5."

The answer is about $\sqrt{365}$, in fact the correct answer is 16.

Let's discuss the intuition, somewhat informally. Take one boy. The expected number of birthdays he shares with one girl is $1/365$.[90] The expected number of birthdays he shares with $k$ girls is $k/365$. So we can say that the expected number of birthday sharing for $k$ boys is $k \times (k/365) = k^2/365$. So if $k = \sqrt{365}$ it it seems that there is a reasonable constant probability that at least one such "sharing" occurs.

Now think of messages as children and of their digests as birthdays. So, if we have two sets $A$ and $B$, each of 16 messages, then with probability at least 0.5, there is at least one $a \in A$ and at least one $b \in B$ such that $a$ and $b$ have the same digest.

In general, see F[Table 11.3, Problem 4], if the universe of digests is of size $N$ and $k = 0.83 \times \sqrt{N}$, then there is a collision (a message from set $A$ has the same digest as a message from set $B$) with probability at least 0.5.

Let us consider a scenario where Mallory will take advantage of such a collision. Alice is a CEO of a company, Mallory is her secretary. Bob is an employee. Bob and Mallory are friends and suspect that Alice is going to demote Bob.

Let us say that the universe of the digests is all the bit strings of 64 bits, that is, using the notation just above, $N = 2^{64}$. Mallory prepares two sets of $2^{32}$ messages as depicted in the following paragraph:

> I, Alice, promote/demote Bob, by this letter/memo so that the organization/enterprise will function better ...

What we have here is a string of letters with 33 pairs of the form "string/string", although we wrote only the first 3 pairs. Set $A$ of messages will start with "I, Alice, promote" and will have all possible choices of selecting one of the other strings from each pair.

Using our example, there will be four messages in $A$:

[89]The standard birthday paradox talks about $k$ people and asking about the probability of any two sharing a birthday and not dividing them into boys and girls.

[90]The probability is $1/365$ that he shares a birthday with this one girl.

> I, Alice, promote Bob, by this letter so that the organization will function better ...
>
> I, Alice, promote Bob, by this letter so that the enterprise will function better ...
>
> I, Alice, promote Bob, by this memo so that the organization will function better ...
>
> I, Alice, promote Bob, by this memo so that the enterprise will function better ...

So there will be $2^{32}$ elements in this set. Set $B$ will be constructed similarly, but will contain message starting with “I, Alice, demote”. Mallory finds a pair $\langle a, b\rangle$, where $a \in A$ and $b \in B$ with the same digest, that is $h(a) = h(b)$ (he can do it with probability greater than 0.5). This may require a lot of work, but can be done well in advance.

Alice decides to demote Bob and tells Mallory to produce the appropriate document. He gives her $b$. She produces $h(s)$, decrypts it with her private key and gets signature $s$.

But this $s$ is also her signature on $a$, as $h(a) = h(b)$ . So now Mallory has a letter $a$ promoting Bob with Alice’s signature on it. And this is computationally not too difficult to do.[91]

Therefore, if we use disgests of length $m$, two messages with the same digest can be produced after trying about $2^{m/2}$ messages.

So, 64 bits is not enough, 160 is considered minimum and even this makes people nervous.

## 14.3 Assuring integrity and authentication of the digest

Let’s look at some more sophisticated ways assuring message integrity and authentication through digests.

Let’s look at Fig. 66. Here, Alice’s chooses some Initialization Vector before computing the digest. So, Mallory cannot precompute the various messages as before, because the digests will depend on the Initialization Vector. For message integrity checking, not only the message and the digest need to be known, but also the Initialization vector. This is not a problem, it can be transmitted also, but it should not be easily predictable, so collisions could not have been precomputed before, as Mallory could not have

[91]And this even under the assumption that the hash function is “perfect,” so nothing other than exhaustive search works to “break” it.

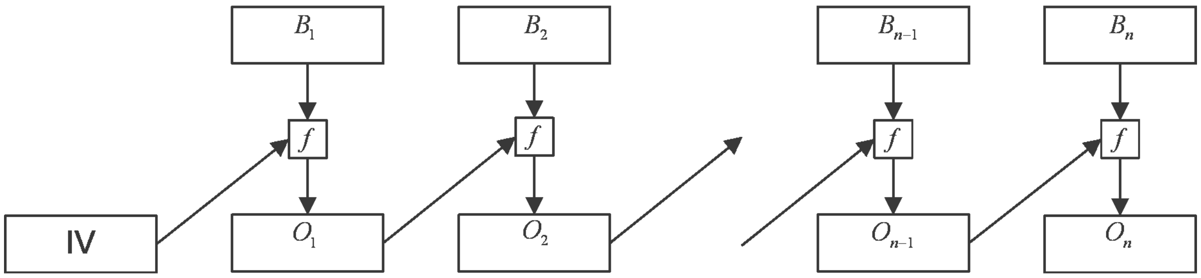


Figure 66: Digest using an initialization vector.

precomputed collisions not knowing how the messages must start—they will start with an unpredictable IV.

But, note that we cannot prevent replay attacks, unless we work a little more. What is replay attack? Replay attacks are important in other contexts, but this is a good place to introduce them. Say the message is from Alice to Bob. Mallory gets hold of this message and makes a copy and let the original one go. A day later he sends another copy of the message to Bob. Bob thinks that he got another (new) message from Alice and acts upon it (such as giving some more money to Mallory). The way to prevent this, perhaps is to use an IV that is really a timestamp.

However, note that there is no proof that Alice is the one who produced the message. Let's now do it, using symmetric key cryptography, by extending the previous discussion.

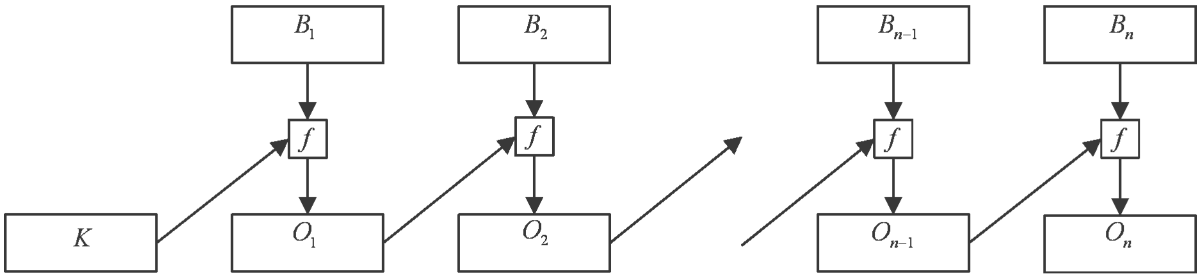


Figure 67: Digest using a key.

1. Look at Fig. 67. We consider messages that Alice sends to Bob. Alice and Bob share a symmetric key.[92] Then only Alice and Bob can create the digest (and of course Bob can confirm that the message came from

[92]Let's not worry about replay attacks here.

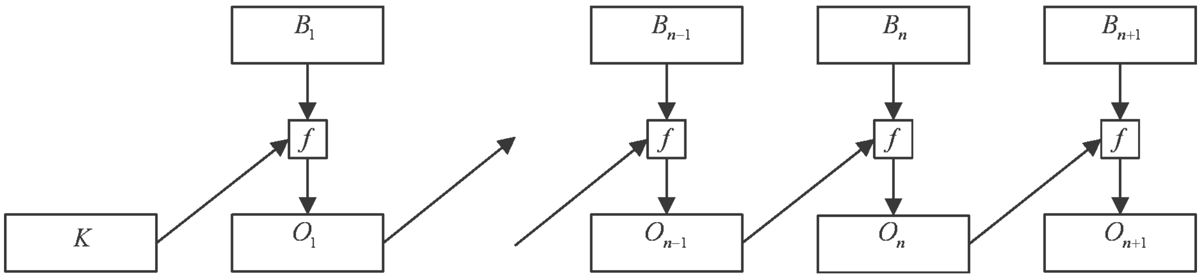


Figure 68: Mallory extends the messages digested by $O_{n+1}$ without knowing the key $K$.

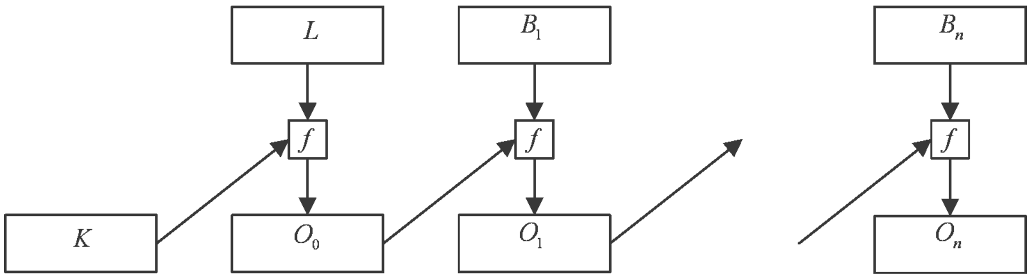


Figure 69: Mallory cannot extend the message as the length, $L$ is specified.

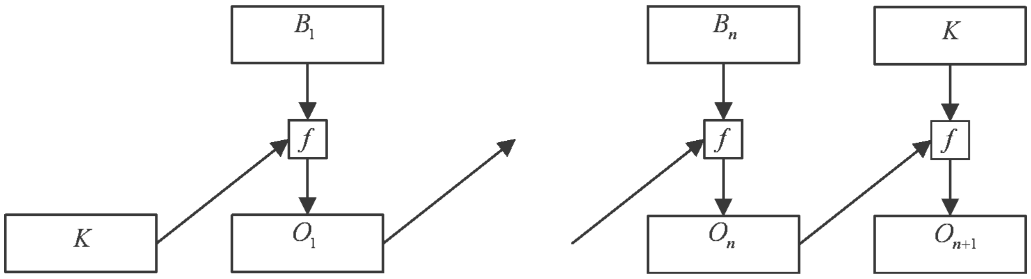


Figure 70: Mallory cannot extend the message as the last block of to be digested is the secret key $K$, which he does not know.

Alice). Of course the key is not sent in the clear. But this (as the previous schemes, though we did not discuss it) is vulnerable to an extension attack.

2. Look at Fig. 68. Here Mallory can trivially extend the message and modify the digest accordingly. There are ways to prevent this.

3. Look at Fig. 69. The message is prepended with $L$, the length of the message. So if Mallory tries to extend the message, the digest will not be correct.

4. Look at Fig. 70. Here is another approach. The key is appended to the message before the digest is computed. So Mallory cannot extend the message and get the correct digest. We have essentially described HMAC. HMAC works with symmetric keys and any hash function.

   A description of HMAC appears in F[354-356] It is slightly different from the one in the official standard http://csrc.nist.gov/publications/drafts/fips_198-1/draft_FIPS-198-1_June-08-2007.pdf.

## 14.4 Key ideas

1. Using one-way hash functions for assuring message integrity and authentication

2. The need for longer digests to prevent birthday paradoxes

3. HMAC

# 15 Discrete log problem and its applications

## 15.1 Discrete log problem

Let us look at the set $\{1, 2, 3, 4, 5, 6\}$ with the operation $\times_7$. (This is really $\mathbb{Z}_7^*$.) Let us look at powers of elements modulo 5. We get a table as shown in Fig. 71.

| $a$ | $a^1$ | $a^2$ | $a^3$ | $a^4$ | $a^5$ | $a^6$ |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 2 | 4 | 1 | 2 | 4 | 1 |
| 3 | 3 | 2 | 6 | 4 | 5 | 1 |
| 4 | 4 | 2 | 1 | 4 | 2 | 1 |
| 5 | 5 | 4 | 6 | 2 | 3 | 1 |
| 6 | 6 | 1 | 6 | 1 | 6 | 1 |

Figure 71: Powers in $\mathbb{Z}_7^*$

Note that powers of 3 give all of the set but powers of 2 do not. We will say that 3 is a *primitive root* of this set; this means that its powers "cycle" through the set.

In general, we can look at $\mathbb{Z}_n^*$ and ask whether it has (at least one) primitive root. The general answer is that it does iff $n$ is one of: $2, 4, p^k, 2 \times p^k$, where $p$ is an odd prime. But we will only look at $\mathbb{Z}_p^*$, with $p \geq 3$, and not other possible $n$.

Returning to our example, both 3 and 5 are primitive roots. We can ask the following questions for any $y \in \mathbb{Z}_7^*$.

1. What is (the unique) $x < n$ such that $y = 3^x$ mod 7? The answer to this is called the discrete log of y to the base 3 modulo 7.[93]

2. What is (the unique) $x < n$ such that $y = 5^x$ mod 7? The answer to this is called the discrete log of y to the base 5 modulo 7.

In general, we can ask this for any $\mathbb{Z}_n^*$ and $g$ that is its primitive root. That is, given $y \in \mathbb{Z}_n^*$, what is the unique $x < n$, such that $y = g^x$ mod $n$.

For a bigger example, consider $n = 19$. 10 is a primitive root of $\mathbb{Z}_n^*$, because computing powers of 10 modulo 19 we get: 10, 5, 12, 6, 3, 11, 15, 17, 18, 9, 14, 7, 13, 16, 8, 4, 2, 1. Note that this sequence does not seem to have a readily understood pattern, so we do not know for instance given 18

[93]Note the analogy with "regular" logs. Just drop the modulus.

(unless we work hard and spend a lot of resources) what was the power to which 10 was raised to get it.

In general, for large $n$ computing discrete logs is considered intractable. So it is a one-way function. Given $x$ it is easy to compute $y = g^x \bmod n$, but given $y$ it is not feasible to compute the corresponding $x$. This is used as a basis for a set of cryptographic primitives.

## 15.2 ElGamal public/private system

We essentially follow F[317-319], using his notation to make reading the textbook simple.

ElGamal is a public/private encryption/decryption system, which is an alternative to RSA.[94] It relies on intractability of computing discrete log. It is considered more secure than RSA in the sense that keys can be made shorter than the ones required by RSA. It can also be implemented using elliptic curves instead of integers with even greater security, that is, even shorter keys are sufficient.. But discussion what elliptic curves are is beyond the scope of our class.[95]

We will only proved a sketch of the protocol.

Bob will generate a public/private key pair.

1. He picks a large prime: $p$
2. He picks a random $d$, $1 \leq d \leq p - 2$
3. He picks $e_1$ a primitive root in $\mathbb{Z}_p^*$
4. He computes $e_2 = e_1^d \bmod p$

His keys are;

1. Public key: $e_1, e_2, p$
2. Private key: $d, p$

Alice wants to send message $P$ to Bob.[96] She does the following:

1. She picks a random $r$, $1 \leq r \leq p - 2$
2. She computes $C_1 = e_1^r \bmod p$ and sends it to Bob

[94] "ElGamal" is the name of the inventor.
[95] These are not ellipses.
[96] Of course, $0 < P < p$.

3. She computes $C_2 = (P \times e_2^r) \bmod p$ and sends it to Bob

Bob wants to recover $P$. He does the following:

1. $P = [C_2 \times (C_1^d)^{-1}] \bmod p$, where of course $-1$ in the exponent means multiplicative inverse modulo $p$.[97]

Why does this work? Let's compute

$$\begin{aligned}
[C_2 \times (C_1^d)^{-1}] \bmod p &= [P \times e_2^r \bmod p] \times [(e_1^r \bmod p)^d]^{-1} \bmod p \\
&= [P \times (e_1^d \bmod p)^r \bmod p] \times [(e_1^r \bmod p)^d]^{-1} \bmod p \\
&= [P \times (e_1^{d\times r} \bmod p) \bmod p] \times (e_1^{r\times d})^{-1} \bmod p \\
&= P \times e_1^{d\times r} \times (e_1^{r\times d})^{-1} \bmod p \\
&= P \times e_1^{d\times r - r\times d} \bmod p \\
&= P \times e_1^0 \bmod p \\
&= P \times 1 \bmod p \\
&= P \bmod p \\
&= P
\end{aligned}$$

## 15.3 Digital Signature Standard (DSS)

This a system used for digital signature only, without enabling encryption/decryption.

The public/private key systems can be used for signing, as we have done before: by decrypting the hash of the message to be signed. So why have a systems just for signing?

The government may prohibit encryption and decryption but may permit signing, so it may be useful to have a system for signing that cannot be used for encryption and decryption, thus making digital signing legal.[98]

We sketch briefly how this works. For more details, you can look at F[405–407]

The following are public:

1. one-way hash $h$

[97]As $p$ is a prime, every $z$, such that $0 < z < p$ has such a multiplicative inverse modulo $p$, which can of course be easily found using Extended Euclidean Algorithm.

[98]But there is something call a "subliminal channel," using signature created following DSS can be used to encrypt small messages—we will see this soon.

2. $q$ prime
3. $p = \alpha \times q + 1$ prime (for some $\alpha$)
4. $g$ such that $g^q \bmod p = 1$

Various computations will be done sometimes modulo $p$ and sometimes modulo $q$. We do not indicate this. There will be various multiplicative inverses, we will not indicate modulo what. Various integers are chosen in various ranges. You can easily find complete specifications or read the details in Forouzan.

Alice chooses her signing (public,private) key pair, by doing the following:

1. Alice chooses random $d$
2. Alice computes $t = g^d$ and publishes it

Alice wants to sign a message $m$. She does:

1. Alice chooses random $S$ (this is necessary for each message separately but is not in any way dependent on the message itself):
2. Alice computes $R = g^S$
3. Alice computes $T = h(m||R)$
4. Alice computes
   $X = d \times T + S$
5. The signature is the pair $\langle T, X \rangle$

Bob gets: $m, T, X$ and verifies:

1. He computes $V = g^X \times t^{-T}$. Note that:

$$\begin{aligned}
V &= g^X \times t^{-T} \\
&= g^X \times (g^d)^{-T} \\
&= g^{X - d \times T} \\
&= g^S \\
&= R
\end{aligned}$$

2. He checks that $h(m||V) = T$

### 15.3.1 Subliminal channel

Using the above scheme, it is possible for Alice to send to Bob short secret messages. To do this she first sends him her $d$, *which she is supposed to keep secret*, in "out of cryptography way."[99]

She now wants to send him a secret. This secret will be used as $S$ (which is supposed to be random, but it is not, it is what she wants to send to Bob). She picks some message $m$. It is not important what it is, for instance: "I am OK."

She goes through the usual procedure.

As before, Bob gets $m, T, X$.

But he knows that $X = d \times T + S$. And as has been told what $d$ was, he just computes:

$$S = X - d \times T$$

## 15.4 Key ideas

1. Primitive roots in $\mathbb{Z}_p^*$
2. Discrete Log
3. ElGamal: using discrete log for public/private encryption system
4. DSA/DSS: using discrete log for digital signatures

[99]Perhaps she told him this $d$ the last time they met.

# 16 Entity authentication

I will mostly use Forouzan and add some material, other than for Zero-Knowledge proofs, which I will do differently.

## 16.1 Introduction

F[416–417]

## 16.2 Passwords

F[417–421]

### 16.2.1 Salt

Note that it really serves like an IV to get different hashes for the same "semantic content."

### 16.2.2 Passphrases

We elaborate on what we already have briefly discussed.

There is a piece of information that must be kept on a disk in a way that nobody can undertand what its "real" value is from the value used for storing it. Examples could be: private key in a public/private cryptosystem or a symmetric key for opening/closing a Truecrypt virtual disk.

Let $S$ be this piece of information. It itself will need to be encrypted with a symmetric key, which needs to be remembered by the user when it is is needed. Say, we use DES.[100]

The user has a one-way hash function $h$. The user remembers a passphrase, a sentence that others would not guess, PassPhrase.

$S$ is not encrypted yet.

The system computes $h$(PassPhrase) and extracts from it 56 bits, adds 8 bits for parity in the right places and gets a 64 DES key, say $K$.

Now, $S$ is encrypted with $K$, producing some $T$. $S$ is erased and $T$ is stored.

When the user needs (or software that the user is running) needs to know $S$, the user inputs PassPhrase, the system computes $h$(PassPhrase) and from it $K$. This $K$ is used to decrypt $T$, producing $S$.

[100]Bad in reality, keys are to short, but easy to remember the length of the keys: 64 bits including parity, 56 bits without parity.

Note, that the passphrase can be changed at any time. The old passphrase is input, $S$ is recovered and then encrypted using a key derived from the new passphrase.

## 16.3 Challenge-Response

We will cover only some of the ones in Forouzan, specifically:

1. Fig. 14.5. Bob sends a random nonce (number once, that is a number that is unlikely to be every used by anybody else, as it was chosen randomly) to prevent replay attacks.
2. Fig. 14.11. We have seen this before.

### 16.3.1 Another protocol for mutual authentication using symmetric keys

Alice and Bob share a symmetric key $K$. They will authenticated themselves to each other using this symmetric key. The various $R$'s are random numbers.

1. Alice $\rightarrow$ Bob

   I am Alice
2. Alice $\leftarrow$ Bob

   $R_1$
3. Alice $\rightarrow$ Bob

   $E(K, R_1)$; Bob verifies
4. Alice $\rightarrow$ Bob

   $R_2$
5. Alice $\leftarrow$ Bob

   $E(K, R_2)$; Alice verifies

Let's try to make it more efficient

1. Alice $\rightarrow$ Bob

   I am Alice, $R_2$
2. Alice $\leftarrow$ Bob

   $R_1$, $E(K, R_2)$; Alice verifies

3. Alice $\rightarrow$ Bob

   $E(K, R_1)$; Bob verifies

This is vulnerable to a reflection attack. Assume that it is OK for Alice to open two sessions with Bob.

In fact, Mallory will open two sessions with Bob and in one of them will convince him he is Alice. We look at the trace. The session in which an event took place is tagged with with a bold integer (1 or 2)

1. Mallory $\rightarrow$ Bob

   **1:** I am Alice, $R_2$

2. Mallory $\leftarrow$ Bob

   **1:** $R_1$, $E(K, R_2)$

3. Mallory $\rightarrow$ Bob

   **2:** I am Alice, $R_1$

4. Mallory $\leftarrow$ Bob

   **2:** $R_3$, $E(K, R_1)$

5. Mallory $\rightarrow$ Bob

   **1:** $E(K, R_1)$; Bob verifies

Mallory breaks session **2** and continues with session **1**.

How to prevent this reflection attack while still using the optimized protocol? Several choices

1. Use different keys known to both. Alice uses $K_{A,B}$ and Bob uses $K_{B,A}$.

2. Alice and both use different challenges (random numbers). Initiator uses odd numbers and responder uses even numbers.[101]

[101] We cannot assign parity "statically," i.e. assigning odd numbers to Alice and even to Bob because then two participants could have the same parity, if say Carol needs to participate also, as then she will have the same parity as either Alice or Bob and there will be a problem if Carol and one of the others need to authenticate each other.

## 16.4 Zero Knowledge

### 16.4.1 Motivation

We want a way in which Peggy (prover) can prove to Victor (verifier) she is Peggy in a way he can verify it.

We want to do it in a way that neither Victor nor Eve can learn anything else. Let's clarify this.

When we look at the way we did it, say in Fig 14.5, Eve can learn what is the encrypted value of $R_{A\text{-}B}$. Whether it is useful to her we do not know, but she did learn something new.

Let's consider another example.

Peggy wants to prove to Victor that she knows the combination to a safe. The safe is empty and open. Victor closes it and leaves the room locking Peggy there (so nobody else can play with the safe in the room other than Peggy). Peggy opens the safe. Victor comes into the room and sees that safe is open. Victor does not know anything new other that the mere fact that Peggy knows the combination.[102]

Cave example from F[428–429]. The important point is that if Peggy does not have the key to the door and they play the game $n$ times, she can succeed in all these times only with probability $1/2^n$.

### 16.4.2 Cut-and-choose and the graph isomorphism problem.

How do we do it in cryptography?

There is a "fact" that only Peggy knows, say $F$. She wants to prove to Victor that she knows this fact.

She creates two "subfacts": $F_1$ and $F_2$, such that:

1. Whoever knows *both* $F_1$ and $F_2$ *provably* knows $F$

2. Knowing only $F_1$ or only $F_2$ does not help in any way to learn anything about $F$

Peggy tells Victor: "I will tell you either $F_1$ or $F_2$, whichever you like, but not both.

Victor chooses 1 or 2 at random. Say, he chooses 1. Then Peggy tells him $F_1$.

[102] We ignore lucky guesses by Peggy.

Now Victor believes that with probability $1/2$ Peggy knew both subfacts. Otherwise, with probability $1 - 1/2 = 1/2$ she did not know $F_1$ (as he picked this subfact at random).[103]

Now she produces two new subfacts of the same fact she claims to know. If again she answers his question, he is now convinced that she is Peggy with probability $1 - 1/4$.

And so on...

We will not discuss this in the class but If $n = p \times q$ a product of two primes, and she knows this factorization, she can convince Victor that she knows the factors of $n$ without giving him any additional information. We will look at a more intuitively understood problem.

We will consider graph isomorphism. We have a (large) graph $G_1$. We permute its vertices and get another graph $G_2$. It anybody sees these two graphs it is difficult (intractable) to find the permutation that permutes the first graph into the second one.

Graphs are represented as the set or vertices and the set of edges, say lexicographically.

Our running example with have the following components:

1. $G_1$, with
   (a) Vertices: 1, 2, 3, 4
   (b) Edges: {1,2}, {1,3}, {1,4}, {3,4}
2. Permutation $\sigma$ of vertices as shown in Fig. 72:
3. This creates $G_2$, with
   (a) Vertices: 1, 2, 3, 4
   (b) Edges: {1,2}, {1,3}, {2,3}, {2,4}

This is pictorially depicted in Fig. 73. But, without the picture it is infeasible (if the graphs are large and nontrivially isomorphic) to figure this out.

Peggy randomly chooses graph $G_1$ of some $n$ vertices and permutation $\sigma$.[104] Computes $G_2$ (that is the list of the edges) and publishes $G_1$ and $G_2$

[103]Technically $\geq 1 - 1/2 = 1/2$, but this is not important. (Perhaps she did not know anything).

[104]We are not discussing how to produce a random graph. Basically, we need to choose edges in some random fashion.

| vertex | $\sigma$(vertex) |
|---|---|
| 1 | 2 |
| 2 | 4 |
| 3 | 3 |
| 4 | 1 |

Figure 72: Permutation $\sigma$ of the vertices of $G_1$

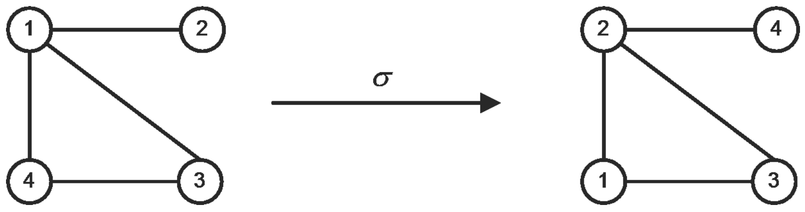


Figure 73: Graphs $G_1$ and $G_2$ and the permutation $\sigma$.

and claims she knows $\sigma$. She is going to prove this to Victor. Note, for the future, that Peggy also knows $\sigma^{-1}$, as this is just the reverse of $\sigma$.

There is a number of rounds. Each round has the following structure:

1. Peggy chooses a random permutation of $\{1, 2, \ldots, n\}$, which is really a permutation of the vertices of $G_1$. This is easy. Let this permutation be called $\pi_1$. It produces some graph $H$. See Figures 74 and 75.

   Note the very important point. Peggy can easily compute the permutation $\pi_2$ that permutes $G_2$ to give $H$. Indeed, if she starts with $G_2$, applies the inverse of $\sigma$, denoted by $\sigma^{-1}$ to it, she gets $G_1$. Then by applying $\pi_1$ to this, she gets $G_2$. In other words, $\pi_2$ is obtained by applying $\sigma^{-1}$ and then $\pi_1$. We see the picture in Fig. 76.

   The permutations $\sigma^{-1}$ and $\pi_2$ are listed in Figures 77 and 78.

2. Victor randomly ask her to produce either $\pi_1$ or $\pi_2$

3. Peggy does this

Let us note a few things.

1. Anybody who knows $G_1$ and $G_2$ can trivially produce the following

   (a) A permutation $\pi_1$ of $n$ vertices which given $G_1$ produces a graph (call it $H_1$) that is isomorphic to $G_1$ under $\pi_1$ [105]

[105] Any permutation of the $n$ vertices will work.

| vertex | $\pi_1$(vertex) |
|---|---|
| 1 | 2 |
| 2 | 4 |
| 3 | 1 |
| 4 | 3 |

Figure 74: Permutation $\pi_1$.

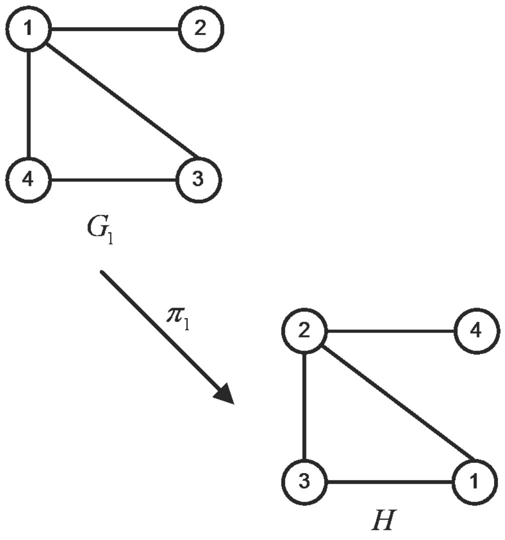


Figure 75: From $G_1$ to $H$.

(b) A permutation $\pi_2$ of $n$ vertices which given $G_2$ produces a graph (call it $H_2$) that is isomorphic to $G_2$ under $\pi_2$

But these $H_1 \neq H_2$ (with extremely high probability). So given either $H_1$ or $H_2$, its "producer" knows how to go back, but *only* from $H_1$ to $G_1$ and *only* from $H_2$ to $G_2$. (But does not know how to go from $H_1$ to $G_2$ or from $H_2$ to $G_1$.)

So if such an actor produces some $H$, it cannot be both $H_1$ and $H_2$. So the actor can only answer *one* of the two questions:

(a) Show $\pi_1$

(b) Show $\pi_2$

2. Anybody who knows *both* $\pi_1$ and $\pi_2$ must also know (by easy) computation permutation $\sigma$.

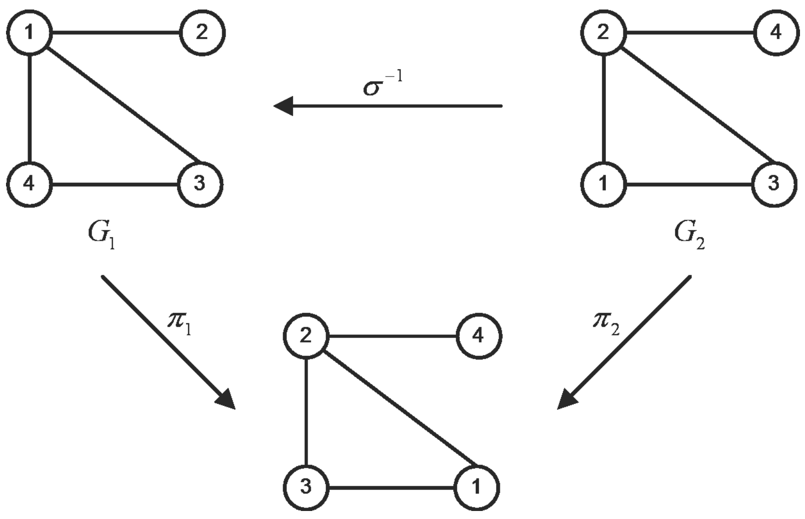


Figure 76: From $\sigma^{-1}$ and $\pi_1$ to $\pi_2$.

| vertex | $\sigma^{-1}$(vertex) |
|---|---|
| 1 | 4 |
| 2 | 1 |
| 3 | 3 |
| 4 | 2 |

Figure 77: Permutation $\sigma^{-1}$.

| vertex | $\pi_2$(vertex) |
|---|---|
| 1 | 3 |
| 2 | 2 |
| 3 | 1 |
| 4 | 4 |

Figure 78: Permutation $\pi_2$.

Indeed, $\sigma$ is obtained by first applying $\pi_1$ and then $\pi_2^{-1}$, the latter is of, course easily computed from $\pi$. This is shown in Fig. 79

Therefore, we now have a procedure by means of which Peggy can authenticate herself to Victor.

They play the game some $k$ times. Each time, she chooses a (different, of course) random $\pi_1$. If she answers correctly the random requests of Victor for either $\pi_1$ or $\pi_2$, he is sure with probability $1 - 1/2^k$ that he is talking to

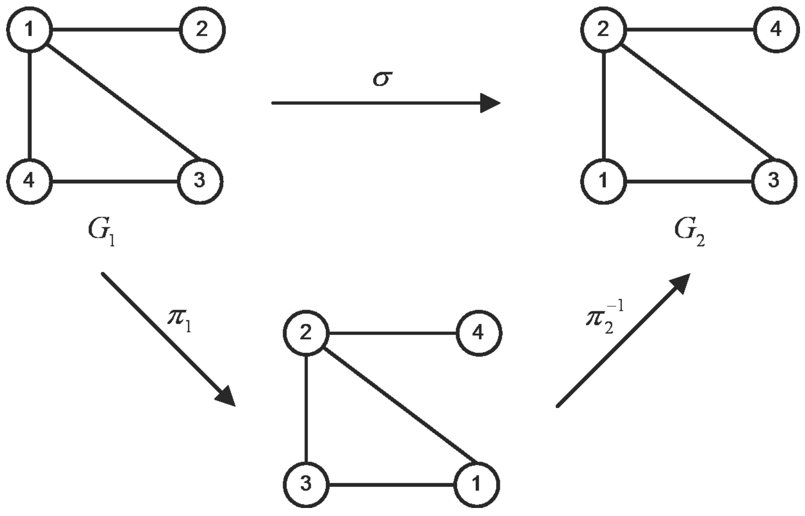


Figure 79: From $\pi_1$ and $\pi_2^{-1}$ to $\sigma$.

Peggy.

Why, because if she can answer *either* question, she must know *both* answers and if she knows both answers it cannot be that she is ignorant of $\sigma$ as she could have easily computed it given $\pi_1$ and $\pi_2$, which she knows.

### 16.4.3 Extension to off-line, non-interactive proofs

The previous procedure was interactive. Let us say that we keep a transcript of what happened, that is the string

$$G_1, G_2, H_1, c_1, \pi_{c_1}, H_2, c_2, \pi_{c_2}, \ldots, H_k, c_k, \pi_{c_k}$$

The results of the $i$th trial are marked with the subscript $i$. Here, $c_i$ is either 1 or 2, $\pi_{c_i}$ the appropriate mapping of either $G_1$ or $G_2$ onto $H_i$ as needed.

Imagine now that somebody brings a transcript of this form. Can we believe that it must have been produced by Peggy? No, anybody can make a transcript of this form.

Let us just look at the first two "trials." Mallory knows $G_1$ and $G_2$. He does the following:

1. He picks $c_1$ any way he likes, say it is 1.

2. He picks $\pi_1$ any way he likes

3. He computes $H_1 = \pi_1(G_1)$
4. He picks $c_2$ any way he likes, say it is 2.
5. He picks $\pi_2$ any way he likes
6. He computes $H_2 = \pi_2(G_2)$

In this wasy he can produce a valid-looking transcript, because he first chose a permutation (the response) and then produced the appropriate graph $H$ (the challenge).

There is a way of describing transcripts that only Alice can produce.

The transcript will have the form

$$G_1, G_2, H_1, H_2, \ldots, H_k, , c_1, c_2, \ldots, c_k, \pi_{c_1}, \pi_{c_2}, \ldots, \pi_{c_k}$$

The key point here is that the challenges come before responses. What needs to be done is to "simulate" Victor, that is to produce challenges that Alice cannot control.

Actually easy. We compute the hash

$$h(G_1, G_2, H_1, H_2, \ldots, H_k)$$

(considering the arguments as one string) and extract the last $k$ bits. Considering each bit $b_i$ as an integer, we define $c_i = b_i + 1$. Only then it is known which $\pi_i$'s are needed. It was unpredictable when the various $H_i$'s were produced.

One can also use if for signatures, to sign $m$, the transcript will be of the form:

$$m, G_1, G_2, H_1, H_2, \ldots, H_k, , c_1, c_2, \ldots, c_k, \pi_{c_1}, \pi_{c_2}, \ldots, \pi_{c_k}$$

and the hash producing the challenges

$$h(m, G_1, G_2, H_1, H_2, \ldots, H_k)$$

# 17 Key management

We look at some parts of F[Chapter 15]. It is easy to read, so we will not spend much time on this topic.

## 17.1 Key distribution center

We just discuss the idea briefly, following Forouzan[bottom of p. 438 – top of p. 439]

## 17.2 Diffie-Hellman

We partially follow Forouzan, Section 15.3

Everybody picks a prime number $p$ and a primitive root (also called generator, the term we have not used) $g$ of $Z_p^*$. To remind you, this means, that as we look at the powers of $g$, we get all the elements: $1, 2, \dots, p-1$.

Alice and Bob each pick a random number, in the range $1, \dots, p-1$.[106] Alice picks $x$ and Bob picks $y$. They will cooperatively generate a secret symmetric key to be used with a symmetric cryptographic system such as DES.

The protocol they use is shown in Figure 15.9 on page 448 in Forouzan. Note that $R_1$ and $R_2$ could be published, so there is no need for Alice and Bob to exchange messages to agree on the symmetric key.[107]

However, as shown in Figure 15.11, this protocol is vulnerable to Mallory's attack.

This can be prevented by Alice and Bob having certificates, so they can use a signing system (such as application of RSA), so they can sign their $R$'s.

This is shown in Figure 15.12.

It is worth briefly to discuss how Diffie-Hellman is used in the email context, using Forouzan's notation.

Bob has published his $R_2 = g^y \bmod p$. Alice, wants to send him a secret message.

She picks a random $x$ and computes $R_1$ and $K$ as before.[108] She uses $K$ to encrypt the email and sends both the encrypted email and $R_1$ to Bob.

[106] 1 and $p-1$ are bad choices as we know, but they will not be chosen (with overwhelming probability).

[107] Generally this is not a good idea, as every "conversation" should have its own encryption key, in case in some way a key can be found (perhaps it was written on a sticky note)

[108] Or she can use the same $x$ with all recipients, not a good idea, better a different key for each email.

# 18 Secret sharing

Alice has a secret string $s$. She wants to give some information to Bob and some to Carol, so together they can reconstruct $s$ but neither can do it alone.

Very easy. She picks random string $r$ of the same length. She gives $r$ to Bob and $r \oplus s$ to Carol. Then can together reconstruct $s$ by computing:

$$r \oplus (r \oplus s) = (r \oplus r) \oplus s = s$$

## 18.1 General case

In this case, we want to have a secret split among an arbitrary number of participants: $n$. Any $m$ of them can reconstruct the secret, but no $m-1$ can do that.

We do everything here using "infinite precision" real numbers. Of course, in practice we need bounded precision arithmetic—just like we did when working with modular arithmetic. This can be done, but we will not discuss how (using some finite structures, fields in this case).

We chose $n$ and $m$, such that $1 \leq m \leq n$.[109] For any string $s$, we can find $n$ string $t_1, t_2, \ldots, t_n$, such that:

1. Given *any* $m$ of the strings from $\{t_1, t_2, \ldots, t_n\}$, there is a unique reconstruction of $s$ from them
2. Given fewer than $m$ such strings, it is impossible to reconstruct $s$ (there is no unique $s$ corresponding to them.

Here is one way of doing it. We will write $s$ as a sequence of bits. Without loss of generality, let us assume that the length of this sequence is divisible by $m$.

Let's split $s$ into blocks of length $m$, and call these blocks: $s_0, s_1, \ldots, s_{m-1}$. Each of these blocks, being a string of bits, naturally defines a non-negative integer. Let us use the notation $s_i$, both for the string of bits and the corresponding integer.

We write a polynomial:

$$y = \sum_{i=0}^{m-1} s_i \times x^i$$

This is a polynomial of degree $m-1$.

[109]The case of $m = 1$ is really trivial because everybody has to know the complete secret.

It is well-known that given $m$ points on the plane, there is at most one polynomial of degree $m-1$ passing through them. For example, given 2 points on the plane, there is at most one polynomial of the form

$$y = \alpha + \beta \times x$$

passing through these points. In fact, there will be exactly one polynomial, unless the two points lie on a line of the form $x = \gamma$.

Let us now consider the example where $n = 3$ and $m = 2$

Our string written as the sequence of bits is $s = 01101000$. Then, written as strings: $s_0 = 0110$, $s_1 = 1000$, and written as integers, $s_0 = 6$ and $s_1 = 8$. Our polynomial is just:

$$y = 6 + 8 \times x$$

Let us pick $n$, that is 3, values of $x$. The choice will not be important, so we may as well pick the integers: 1, 2, 3. Let us compute $y(x)$:

$$\begin{aligned} y(1) =& 14 \\ y(2) =& 22 \\ y(3) =& 30 \end{aligned}$$

Look at Figure 80

We now have 3 points on the line: $t_1 = \langle 1, 14 \rangle$, $t_2 = \langle 2, 22 \rangle$, and $t_3 = \langle 3, 30 \rangle$.

Any two of these three points specify exactly one straight line, actually our line (polynomial of degree 1). So if we give one point to a person, any two of them can find the intersection and therefore reconstruct the string $s$

For example if they together know $\langle 1, 14 \rangle$ and $\langle 2, 22 \rangle$, the will solve the system of linear equations:

$$\begin{aligned} 14 =& s_0 + s_1 \times 1 \\ 22 =& s_0 + s_1 \times 2 \end{aligned}$$

A single point is not sufficient as an infinite number of lines pass through it.

In general, given a polynomial of degree $m-1$, it is fully determined by any $m$ pairs of the form: $\langle x_i, y_i \rangle$, where $y_i = y(x_i)$. So given these $m$ points/pairs we need to solve for the pieces $s_0, \ldots, s_{m-1}$

We write $m$ linear equations for $i = 0, 1, \ldots, m-1$:

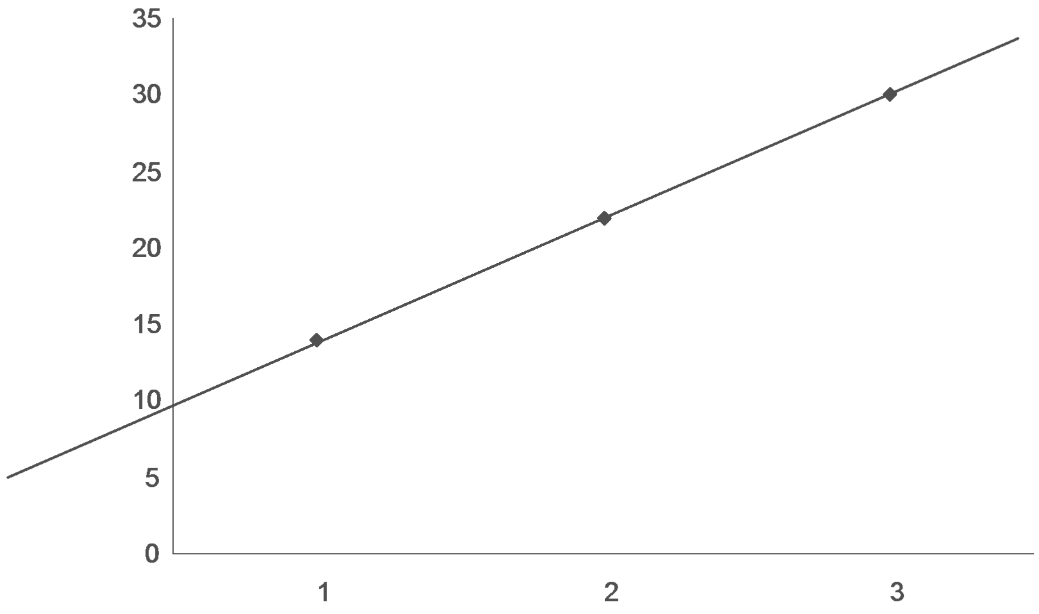


Figure 80: Three points on a straight line.

$$y_i = \sum_{j=0}^{m-1} s_i \times x_i^j$$

and solve this system of $m$ equations in $m$ variables to get $s_0, s_1, \ldots, s_{m-1}$.

So, for the general problem we have, we write the appropriate polynomial of degree $m-1$ with the coefficients being the "pieces" of the string $s$, compute its value in $n$ points, say $1, 2, \ldots, n$, and this is how we get the needed $t_i$'s

There is another way, quite similar. Look at Figure 81.

Again, let $n = 3$ and $m = 2$. Take the point $\langle s_0, s_1 \rangle$ in 2-dimensional space.

Pick any 3 lines passing thought this point: $y = \alpha_i + \beta_i \times x$, for $i = 1, 2, 3$. $t_i = \langle \alpha_i, \beta_i \rangle$

Any two lines from the three above intersect at exactly the point $\langle s_0, s_1 \rangle$, producing $s$. So given any two of the $t_i$'s allows reconstruction of the point.

This can be easily generalized to any $m$ and $n$

Useful application: hiding a treasure in a desert. Give one person the longitude and the other the latitude. Here, $n = m = 2$.

In practice:

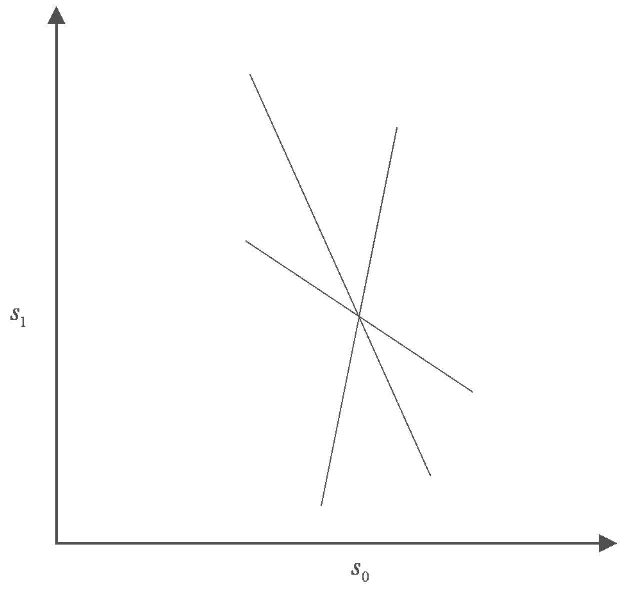


Figure 81: Three lines intersecting at one point.

**Theorem 16.** Given string $s$, it is possible to find $n$ strings each of length length$(s)/m$, where length$(s)$ is the number of bits in $s$.

such that from any $m$ of them, $s$ can be (uniquely) reconstructed. (Without loss of generality, as we can always pad, the length of $s$ is divisible by $m$.)

This is optimal, as we reconstruct length$(s)$ bits out of length$(s)$ bits.

□

This is related (closely) to error-correcting codes.

## 19 Secure Socket Layer (SSL)

This is what is going on when you see a padlock.

Forouzan provides an extremely detailed description of this protocol. I will, instead, tell you the key ideas behind what is going on.

First, just for the record, here is the *handshake protocol* between a client and a server (some of the messages are optional, for instance the client may not have a certificate):

1. Phase 1
   (a) Client $\rightarrow$ Server
   client_hello
   (b) Client $\leftarrow$ Server
   server_hello
2. Phase 2
   (a) Client $\leftarrow$ Server
   certificate
   (b) Client $\leftarrow$ Server
   server_key_exchange
   (c) Client $\leftarrow$ Server
   certificate_request
   (d) Client $\leftarrow$ Server
   server_hello_done
3. Phase 3
   (a) Client $\rightarrow$ Server
   certificate
   (b) Client $\rightarrow$ Server
   client_key_exchange
   (c) Client $\rightarrow$ Server
   certificate_verify
4. Phase 4

   (a) Client → Server
       change_cipher_spec
   (b) Client → Server
       finished
   (c) Client ← Server
       change_cipher_spec
   (d) Client ← Server
       finished

Let us now examine the key actions that take place, not necessarily in the exact order in which they take place (we combined various messages into one action, sometimes):

1. Client → Server
   (a) I want to talk
   (b) here is the highest version of SSL protocol I understand
   (c) here are the cryptographic methods that I can handle: public/private, symmetric, hash [110] [111]
   (d) here is my random number (nonce): $R_{\text{Client}}$.
2. Client ← Server
   (a) here is the version of SSL we will use
   (b) here is my certificate (containing my public key)
   (c) here is my choice of cryptographic methods from the ones you can handle;[112]
   (d) here is my random number (nonce): $R_{\text{Server}}$.
3. Client → Server
   (a) I am sending you a random secret $S$ encrypted with your public key: $E^{\text{RSA}}(e^{\text{RSA}}_{\text{Server}}, S)$

[110]For instance, the client can specify that it can handle the following symmetric cryptosystems: 3DES and AES.
[111]These are also called "ciphers."
[112]For instance, RSA, 3DES, Whirlpool.

   (b) I am computing for myself our shared secret $K = f(R_{\text{Client}}, R_{\text{Server}}, S)$ where $f$ is a standard function [113] [114]
   (c) a (keyed) hash of the string CLNT||handshakeMessages [115]

4. Client ← Server
   (a) I am computing for myself our shared secret $K = f(R_{\text{Client}}, R_{\text{Server}}, S)$ where $f$ is a standard function [116] [117]
   (b) I am sending you a (keyed) hash of the string SRVR||handshakeMessages;[118]

Let us note:

1. If the handshake messages have been tampered with, this will be detected by means of the keyed hash.
2. The two random numbers prevent replay attacks: an old session cannot be reused.
3. The client authenticates the server in an oblique way: the server will get the same shared pre-secret $S$ only if it can decrypt the message $E^{\text{RSA}}(e^{\text{RSA}}_{\text{Server}}, S)$, and it can do this only if it knows the private key of the "alleged" server.
4. The shared secret key $K$ is used to generate the various secrets needed, IV (Initialization Vector), symmetric encryption/decryption key, and symmetric signing key,[119] each for each of the two participants, in fact 6 total.

Now they can communicate. It is is now easy to see how they do it. They all know all the 6 secrets.

If the client wants to talk to the server it uses its own IV, encrypting key, and signing key. The server knows the client's secrets so it can decrypt and authenticate what it gets.

[113] All the needed "secrets," i.e., keys are computed in a predictable way from this $K$
[114] This actually does not need to be reported, this part of the protocol.
[115] These are messages exchanged so far.
[116] The server gets $S$ by decrypting what the client sent it previously.
[117] This actually does not need to be reported, this part of the protocol.
[118] Note that the server and the client have slightly different strings hashed, so the actual hashes will be very different
[119] Recall how we used a symmetric key in the context of keyed hashed functions to authenticate a message.

If the server wants to talk to the client it uses its own IV, encrypting key, and signing key. The client knows the server's secrets so it can decrypt and authenticate what it gets.

# 20 Digital cash

Cash, as opposed to credit cards, wire transfers, or checks, has some good properties (anonymity) and some bad properties (can be forged with the forgerer not traceable, some of this holds true also for checks and other means of payments). Digital cash has anonymity and cannot be forged. We will learn how to make and use digital cash.

Instead of showing it all at once, we will produce a series of "implementations," with the last one being a correct/complete one. In the middle, we will learn a few interesting things.

Bob is a bank. He issues "bills" of \$1. The bill is essentially a nonce (number once), which is the "serial number"; and a signature by Bob. We trust Bob, he is like Trent.

Alice comes to Bob and gives him a real \$1 bill. We assume that he knows who she is.[120] He gives her some bits. Whenever Alice wants to buy something from say Carol, she gives her the bits. The vendor gives these bits (perhaps something else too) to Bob and gets from Bob a real \$1 bill.

Bob has some public/private key system for signing, let's assume RSA. He has, of course, the corresponding public private keys, based on $n, e, d$. He uses these only for his money issuing activities and nothing else. (He can, of course, have another key pair for other activities.)

## 20.1 Digital cash, take 1

This is really a review. We have seen this before.

Bob chooses a nonce (as usual randomly) $x < n$.[121] He gives Alice the pair $\langle x, x^d \bmod n\rangle$. The second part of the pair is the signature of Bob on the first part. Carol can check the validity by checking that:

$$(x^d \bmod n)^e \bmod n = x$$

That is, encrypting the second part produces the first part.

Of course, this can be forged by Mallory. Anybody can pick $y < n$ and compute $y^e \bmod n$. Then, of course, the pair $\langle y^e \bmod n, y\rangle$ looks like a valid bill.

[120] Reasonable assumption, as she very likely actually wires the money to his account in some way.

[121] Alice could have picked $x$, but Bob has to verify that he has never used $x$ before in what follows.

## 20.2 Digital cash, take 2

This is really a review. We have seen the signing before.

Everybody has a well-known one-way hash function $h$. Bob chooses a nonce (as previously) $x < n$. He gives Alice the pair $\langle x, h^d(x) \bmod n \rangle$. The second part of the pair is the signature of Bob on the first part. Carol can check the validity by checking that:

$$((h^d(x) \bmod n)^e \bmod n = h(x)$$

That is, encrypting the second part produces the hash of the first part.

This cannot be forged in the following sense: If this bill is given back to Bob twice, he knows that one of them is a forgery, as it is really a copy. But he does not know which is a forgery, maybe both, so he really does not know what to do, who, if any should be paid by him with a real \$1 bill.

Also, there is no anonymity. When Carol presents the bits to Bob, he knows that Carol got them from Alice as he knows to whom he gave the pair.

## 20.3 Digital cash, take 3, "blind signing" to preserve anonymity

Here we will learn an interesting protocol: *blind signing.* This corresponds to signing a sealed envelope.

Alice picks two random numbers $x, z < n$. $z$ is relatively prime with $n$. She also computes the multiplicative inverse of $z$ modulo $n$, let's denote it by $z^{-1}$.

She also computes $y = h(x) \times z^e \bmod n$ and sends it to Bob. *Bob will not sign it but will do what signing does without taking the hash first.* We will see soon that this will not cause a problem So Bob gives her back the pair $\langle y, y^d \bmod n \rangle$. This is in fact:

$$\begin{aligned}(h(x) \times z^e \bmod n)^d \bmod n =&(h^d(x) \bmod n) \times (z^{ed} \bmod n) \bmod n \\ =&(h^d(x) \bmod n) \times (z) \bmod n\end{aligned}$$

Alice now multiplies this by $z^{-1}$ and gets

$$h^d(x) \bmod n$$

So she now has Bob's signature on $x$, that is the hash of $x$ decrypted with Bob's private key and nobody can produce this other than Bob.

Alice buys something from Carol by giving her the pair:

$$\langle x, h(x)^d \bmod n \rangle$$

Carol checks that this is a valid \$1, by checking that the second part is Bob's signature on the first part, and gives it to Bob to get a real \$1. Bob will give her real money if he has never paid it before.[122]

We have anonymity. Bob does not know to whom he has issued the pair. *This was blind signing.*

But, what if Alice cheats and she spent the money already with Dona?

## 20.4 Digital cash, take 4, "cut and choose" to prevent copying money

Let $f$ and $g$ be one-way hash-functions of two variables.[123] Let $k$ be some integer (not very large, we will discuss later).

Alice picks some random (everywhere, numbers are smaller than $n$ as needed).

$$\begin{aligned}
&v_1, \ldots, v_k \\
&a_1, \ldots, a_k \\
&c_1, \ldots, c_k \\
&d_1, \ldots, d_k \\
&z_1, \ldots, z_k \qquad \text{where each } z_i \text{ is relatively prime with } n
\end{aligned}$$

Alice$||v_i$ is the concatenation of the string "Alice" with the integer $v_i$.

Alice computes

$$x_i = g(a_i, c_i) \quad \text{and} \quad y_i = g(a_i \oplus \text{Alice}||v_i, d_i) \qquad \text{for} \quad i = 1, \ldots, k$$

Note that this looks like secret sharing, as the string Alice$||v_i$ is recoverable from $a_i$ and $a_i \oplus$ Alice$||v_i$, but not from one of them.

Alice computes and sends to Bob:

$$f(x_i, y_i) z_i^e \bmod n \qquad \text{for} \quad i = 1, \ldots, k$$

[122]Some people do not consider real money anything that is not some fixed amount of gold, so we could have replace a real \$1 bill by a grain of gold.

[123]Of course, these could be one-way functions with the two variables coded into one variable by concatenation with a well-known separator between them. So we really need only one function $h$.

He blindly signs these, as before, by decrypting them.

Alice gets rid of $z_i$'s as before, getting

$$\alpha_i = f^d(x_i, y_i) \bmod n \qquad \text{for} \quad i = 1, \ldots, k$$

She sends these to Carol. Carol randomly partitions $\{1, \ldots k\}$ into two equal sized sets,[124] and ask different question for $i$'s in the two sets.[125]

1. If $i$ is in set 1, she asks Alice to produce: $a_i, c_i, y_i$. She computes $g(a_i, c_i)$. Note that this is $x_i$. She then checks that

$$f(x_i, y_i) = \alpha_i^e \bmod n$$

2. If $i$ is in set 2, she asks Alice to produce: $a_i \oplus \text{Alice}||v_i, d_i, x_i$. She computes $g(a_i \oplus \text{Alice}||v_i, d_i)$. Note that this is $y_i$. She then checks that

$$f(x_i, y_i) = \alpha_i^e \bmod n$$

If everything is verified, then Alice sent Bob's signature on the right strings. Carol shows everything to Bob, and he pays unless he has a proof that he has already paid. He does not, so he pays.

Of course, if Carol resubmits the same strings, he has a proof that he already got those strings—same if Carol gave these strings to Donna. But what if Alice now spends the same \$1 with Frank?

Frank ask her to do exactly the same thing that Carol did. But he picks his own partition of $\{1, \ldots k\}$.

If $k$ is large enough (does not have to be very large) then it is highly likely that for some $i$ Carol had it in set 1 and Frank in set 2 (or vice versa). So Bob received together from Carol and Frank for that $i$

$$a_i, c_i, y_i, a_i \oplus \text{Alice}||v_i, d_i, x_i$$

He computes

$$a_i \oplus (a_i \oplus \text{Alice}||v_i) = \text{Alice}||v_i$$

and recovers "Alice." So he knows she is a forgerer and also has a proof that he has already paid.

[124] Of course we can assume that $k$ is even, or we can have the two sets varying by 1 in size.

[125] Note this is similar to what we did in Zero Knowledge proofs.

However, what if Alice, when submitting her strings to Bob at the beginning, replaced her name with George, Henry, Irene,.... After, he did not see the strings he signed as she blinded them first.

Things fall apart.

## 20.5 Digital cash, take 5, "cut and choose" to prevent impersonation

We need to make sure that Bob knows that Alice put her name correctly in the strings before hashing.

Alice prepares the strings as before, but with the parameter $2k$ instead of $k$. Bob gets $2k$ strings of the form

$$f(x_i, y_i)z_i^e \bmod n \qquad \text{for} \quad i = 1, \dots, 2k$$

He randomly picks a set consisting of half of them ($k$ out of $2k$). For each $i$ that was picked, he asks Alice to show him everything:

$$v_i, a_i, c_i, d_i, z_i$$

From these he can verify that he got the right

$$f(x_i, y_i)z_i^e \bmod n \qquad \text{for} \quad i \quad \text{in the chosen set}$$

If everything is OK, he throws these out and blindly signs the other ones and gives the signatures to Alice.

Alice can still smuggle in some small number perhaps of bad inputs, but the intersection of the two sets that Carol and Donna picked is quite large and there are many strings there with Alice's name.

# 21 Identity-based encryption

We will cover this very informally.

The following is true, though we cannot discuss the details. Given some string, say $s$, it is possible to find knowing only this string a public private key pair $(e, d)$ in a deterministic way (no random numbers) using a known algorithm. Therefore anybody who knows this algorithm can do it. We will make this clearer in a sequence of "implementations."

## 21.1 Identity-based encryption: take 1

There exists a particular algorithm, or using our terminology, a particular machine $\mathcal{M}$. This machine, for any name, such as Alice,[126] produces two strings:

1. $e^{\text{ID}}_{\text{Alice}}$
2. $d^{\text{ID}}_{\text{Alice}}$

such that the two strings can serve as (public,private) key pair in some (public,private) key system. The superscript will remind us that this is "identity-based encryption."

Now, anybody who knows the algorithm can figure out $e^{\text{ID}}_{\text{Alice}}$ and $d^{\text{ID}}_{\text{Alice}}$ therefore can send Alice an encrypted message. The problem is, of course, that anybody who knows the algorithm can also figure out $d^{\text{ID}}_{\text{Alice}}$ and therefore anybody (including Eve) can decrypt messages sent to Alice.

## 21.2 Identity-based encryption: take 2

The algorithm is broken into two subalgorithms: $\mathcal{M}^{\text{E}}$ and $\mathcal{M}^{\text{D}}$. The first one is published, the second one is only known to Trent. Trent computes $d^{\text{ID}}_{\text{Alice}}$ and gives it to Alice. So only Alice and Trent can decrypt messages sent to Alice.

There are two problems, which we will discuss in turn:

1. We do not like secret machines, we like public machines controlled (if appropriate) by keys (Kerckhoff's law)
2. We do not want a single Trent in the whole universe to be able to read everybody's email. Note, that the way we managed certificates in RSA, Trent did not know private keys, but certified people's public keys.

[126]This has to be a unique identifier for the person, probably best is an email address, because this is known to whoever wants to send email to Alice

## 21.3 Identity-based encryption: take 3

The machines for generating public/private keys are publicly known, but they are controlled by *master keys.* So Trent has some kind of master key pair $(e_{\text{Trent}}^{\text{Master}}, d_{\text{Trent}}^{\text{Master}})$. There are two machines, both publicly known: $\mathcal{M}^{\text{E}}$ and $\mathcal{M}^{\text{D}}$.

Given a string, say Alice, and the appropriate master keys, the machines produce the following keys unique to Alice:

1. $e_{\text{Alice}}^{\text{ID}} = \mathcal{M}^{\text{E}}(e_{\text{Trent}}^{\text{Master}}, \text{Alice})$
2. $d_{\text{Alice}}^{\text{ID}} = \mathcal{M}^{\text{D}}(d_{\text{Trent}}^{\text{Master}}, \text{Alice})$

$e_{\text{Trent}}^{\text{Master}}$ is public, so anybody can compute $e_{\text{Alice}}^{\text{ID}}$. $d_{\text{Trent}}^{\text{Master}}$ is private, so only Trent can compute $d_{\text{Alice}}^{\text{ID}}$ and give it to Alice.

## 21.4 Identity-based encryption: take 4

Every organization has its own Trent, with its own master keys.

So, anybody can send encrypted email to anybody by computing the public key of the recipient (as long as the sender knows the master key for producing public keys of the recipient's organization's Trent. But only Trent of that organization can generate the corresponding private key of the recipient and only those to whom this private key was given can decrypt the emails.

Note, that the organization can read all encrypted email sent to its employees.[127]

Note, that if the set of organization's employees is static, its Trent can throw out its $d_{\text{Trent}}^{\text{Master}}$ once every employee was given his or her private key.

[127]In fact, for legal reasons, organizations may be required to be able to produce all emails, so in future this might be the preferred way for organizations to manage encryption.

## 22 Corrections and additions

**Notation** [x,y] means: page x line y. y could be positive (count forward from the top of the page, starting with $+1$) or negative (start counting backwards from the bottom of the page, starting with $-1$).

## List of Figures

# Contents