Block ciphers (and DES in particular)

We now introduce block ciphers at the example of **DES** (short for data encryption standard).

This sketch only provides an overview but does not include all details. See Chapter 4 in our book for these internals and detailed diagrams.

DES was the first public cryptosystem. While a public standard, the design decisions have been kept secret.

1974: proposed by IBM (lead by Horst Feistel; Lucifer) with input from NSA (key size reduced from 128 to 56 bits)
1976–2000: US national standard (broken by exhaustive search in 1997)
2000: replaced with AES (Rijndael) by NIST; however, **3DES** still considered secure (more later)

Why was the design secret? For many years, a particular mystery about DES was the choice of the S-boxes. Much later, in 1990, Biham and Shamir discovered **differential cryptanalysis**, a general method for breaking block ciphers. Surprisingly, it turned out that the particular choice of S-boxes made DES rather resistant against that attack. Indeed, as confirmed later, the IBM researchers had already discovered and anticipated that attack in 1990, but were asked by the NSA to keep it secret (it was a powerful weapon against other cryptosystems).

https://en.wikipedia.org/wiki/Data_Encryption_Standard

Comment. As our discussion will show, DES was designed to be implemented in hardware.

General principles of block cipher design

A block cipher takes a plaintext block of, say, B bits and encrypts it into a ciphertext block of B bits.

For instance, for DES, $B = 64$: 64 bit blocks are encrypted to 64 bit blocks.

For now, we will just focus on encrypting a single block.

However, we will need to talk about how to use a block cipher to encrypt longer plaintexts that need to be broken into many blocks (it is generally a bad idea to individually and independently encrypt each block).

The design of a block cipher is almost an art, but there are two guiding principles due to Claude Shannon, the father of information theory:

- confusion

refers to making the relationship between the ciphertext and the key as complex and involved as possible

(for instance, changing one bit of the key should change the ciphertext completely)

For instance. In DES, confusion is increased by the S-box substitutions. These are the only nonlinear part of DES. Without them, DES would be easily broken with linear algebra.

- diffusion

refers to dissipating the statistical structure of plaintext over the bulk of ciphertext

(for instance, changing one bit of the plaintext should change the ciphertext completely; likewise, changing one bit of the ciphertext should change the plaintext completely)

For instance. In DES, diffusion is increased by the E-box and P-box permutations.

Example 108. The classical substitution cipher provides only confusion.

Diffusion is completely missing. Changing bits of the plaintext only changes corresponding parts of the ciphertext. That's why frequency analysis can break these ciphers so easily.

Typical block ciphers are built by iteration, and consist of several **rounds**. Each round should have steps to increase both confusion and diffusion.

- **(key expansion)** First, we need to expand key k into several **round keys** $k_1, k_2, ..., k_n$ (n rounds).
  For instance. For DES, each round key k_i has 48 bits, which are drawn from the 56 bit DES key k in such a way that each bit of k shows up in about 14 of the 16 rounds.
- **(round functions)** Then, the message m is encrypted successively with R_{k_1}, R_{k_2}, ..., R_{k_n} to obtain c in the end.

$$m \to \boxed{R_{k_1}} \to \boxed{R_{k_2}} \to \ldots \to \boxed{R_{k_n}} \to c$$

  Each R_k is called a **round function**.
  For instance. For DES, there are $n = 16$ rounds; for AES-128, there are $n = 10$ rounds

A specific block cipher now needs specific algorithms for key expansion and round functions.

For DES, 16 rounds are used, which are identical in functionality but use different round keys k_i.

There is one additional, cryptographically irrelevant, step for DES: namely, there is a (fixed) **initial permutation** IP, which shuffles the bits of m before being sent to R_{k_1}. Similarly, the output of $R_{k_{16}}$ is shuffled with IP^{-1}, the inverse permutation, to produce c. (For the exact permutation see Chapter 4.4 in our book.)
Why? When implemented in hardware, this permutation does not cost any work, since it is just a wiring of the bits. In fact, the permutation somehow simplified the electrical engineering in the chips of the 70s.

A block cipher design: Feistel ciphers

Many ciphers, including DES (but not AES) are Feistel ciphers. This means that the encryption functions R_{k_i} are of a special format. The crucial ingredient is a **round function** $f_{k_i}(x)$.

This round function can be **any** function, such that x and $f_{k_i}(x)$ have the same size in bits (though only good choices will provide security). Also, several different round functions can be used for the different rounds.

To encrypt m using R_{k_i} (for DES, m is 64 bits and the round key k_i is 48 bits):

- Split the plaintext m into two halves (L_0, R_0) (for DES, each half is 32 bits).
- $L_1 = R_0$
  $R_1 = L_0 \oplus f_{k_i}(R_0)$
- Then, $R_{k_i}(m)$ is (L_1, R_1).

Example 109. How to decrypt one round? That is, how to obtain (L_0, R_0) from (L_1, R_1)?

Solution. First, $R_0 = L_1$. Then, $L_0 = R_1 \oplus f_{k_i}(R_0)$.
Important comment. In particular, we can take any round function f in the sense that we obtain some cipher, which can actually be decrypted (however, most choices for f will be insecure; see example below).
Comment. In hardware, the circuit for decryption is the same as for encryption, just reversed.

Example 110. What happens if we choose $f_{k_i}(R) = 0$ as the round function?

Solution. In that case, we are just swapping left and right half. No security whatsoever.

To finish the description of DES, we need to specify $f_{k_i}(R)$, where R is 32 bits and k_i is 48 bits.

We did that in class, but do not reproduce the description and diagrams here. See Chapter 4.4 of our book.
The crucial ingredients are an E-box (expansion), eight S-boxes (substitution) and a P-box (permutation).

Further comments on DES

The S-boxes $S_1, S_2, \ldots, S_8$ are lookup tables (for each 6 bit input, they specify a 4 bit output).

- They have been carefully designed.
  For instance, their design already anticipated and protected against differential cryptanalysis (which wasn't publicly known at the time).
- On the other hand, they do not follow any simple rule. In particular, they must not be linear (or close to it). If they were, DES would be entirely insecure.
  [Slightly more specifically, if the S-boxes were linear, then the encryption map $m \mapsto c$ would be linear. In the usual spirit of linear algebra, a few (m, c) pairs would then suffice to recover the key.]
- They are also designed so that if one bit is changed in the input, then at least 2 bits of the output change.
  Important consequence. Go through one application of the round function $f_{k_i}(R)$, and convince yourself that flipping one bit of R has the effect of flipping at least two bits of $f_{k_i}(R)$. Repeating this for 16 rounds, you can see how the goal of diffusion seems to be achieved: changing one bit of the plaintext should change the ciphertext completely.

Example 111. Sometimes it is stated that DES works with a 64 bit key size. In that case, every 8th bit is a parity bit, but the algorithm really operates with 56 bit keys.

Comment. Apparently, the NSA was interested in strengthening DES against any attack (recall that developments like differential cryptanalysis were foreseen) except brute-force. Indeed, the NSA seems to have pushed for a key size of 48 bits versus proposed 64 bits, and the result was a compromise for 56 bits.

Example 112. If DES is insecure because of its 56 bit key size, why not just increase that?

Solution. DES was designed specifically for that key size. Increasing it necessitates a completely new analysis on how to choose the S-boxes and so on.

On the other hand. See the upcoming discussion of 3DES for how to leverage the original DES to increase the key size.

However. With the advent of powerful successors like AES there are very few reasons to use 3DES for new cryptosystems. (One slight advantage of 3DES is its particular small footprint in hardware implementations.)

Example 113. Can we (easily) break DES if we know one of the round keys?

Solution. Absolutely! Recall that each round key is 48 bits taken from the overall 56 bit DES key. Hence, we know all but 8 bits of the key. We just need to brute-force these $2^8 = 256$ many possibilities.

Example 114. To (naively) brute-force DES, how much data must we encrypt?

Solution. By brute-forcing, we mean that, given a pair of 64-bit blocks m, c, we go through all 2^{56} possibilities (DES uses 56-bit keys) for k and look for k such that $E_k(m) = c$? We need to encrypt 2^{56} times 64 bits.

This is $2^{56} \cdot 8 = 2^{59}$ byte, or 512 pebibyte (binary analog of petabyte) or 576 petabyte (since $2^{59} \approx 5.76 \cdot 10^{17}$).

How long will this take? Of course, this depends on your machine. Assume we are able to encrypt 1 GB/sec. Then, this will take us about $5.76 \cdot 10^8$ sec, or about 18.3 years.

Of course, such a brute-force attack can be fully parallelized to quickly bring this number down to less than an hour for a powerful attacker. Also, the attack can be sped up considerably by careful design (like early aborts).

For comparison. Though mostly of theoretical value, for DES, some possibilities for attacks better than brute-force are known: for instance, as of 2008, linear cryptanalysis can mount an attack with 2^{43} known plaintexts in about 2^{40} (instead of 2^{56}) steps.

Example 115. (bonus!) Using DES, are there blocks m, c such that $E_k(m) = c$ for more than one key k?

I don't know the answer and couldn't find it easily. Maybe you are more skilled?

Example 116. (3DES) A simple approach to increasing the key size of DES, without the need to design and analyze a new block cipher, is **3DES**. It consists of three applications of DES to each block and is still considered secure.

$$c = E_{k_3}(D_{k_2}(E_{k_1}(m)))$$

The 3DES standard allows three keying options:

- k_1, k_2, k_3 independent keys: $3 \times 56 = 168$ key size, but effective key size is 112
- $k_1 = k_3$: $2 \times 56 = 112$ key size, effective key size is stated as 80 by NIST
- $k_1 = k_2 = k_3$: this is just the usual DES, and provides backwards compatibility (which is a major reason for making the middle step a decryption instead of another encryption).

Comment. The reason for the reduced effective key sizes is the meet-in-the-middle attack. It is also the reason why something like 2DES is not used. See next example!

Comment. NIST approved 3DES until 2030 for sensitive government data.

Example 117. (no 2DES) Explain why "2DES" does not really provide extra security over DES.

Solution. Let's denote DES encryption with E_k and decryption with D_k. The keys k are 56 bits.

Then, 2DES encrypts according to $c = E_{k_2}(E_{k_1}(m))$. The key size of 2DES is $56 + 56 = 112$ bits.

- A brute-force attack would go through all possibilities for pairs (k_1, k_2), of which there is $2^{56} \cdot 2^{56} = 2^{112}$, to check whether $c = E_{k_2}(E_{k_1}(m))$. That requires 2^{112} DES computations.
- On the other hand, note that $c = E_{k_2}(E_{k_1}(m))$ is equivalent to $D_{k_2}(c) = E_{k_1}(m)$.

  Assuming sufficient memory, we first go through all 2^{56} keys k_2 and store the values $D_{k_2}(c)$ in a lookup table.

  We then go through all 2^{56} keys k_1, compute $E_{k_1}(m)$ and see if we have stored that value before. (Even though this is a huge table, the cost for checking whether an element is in the table can be disregarded; thanks to the magic of hash tables!)

  Comment. In this second step, we see that m and c should be more than one block (otherwise we get too many candidate keys $k = (k_1, k_2)$).

  The total number of DES computations to break 2DES therefore is $2^{56} + 2^{56} = 2^{57}$, which is hardly more than for breaking DES!

Armin Straub
straub@southalabama.edu

This is known as a meet-in-the-middle attack.

https://en.wikipedia.org/wiki/Meet-in-the-middle_attack

Comment. The price to pay is that this attack also requires memory for storing This sort of approach is referred to as a time-memory trade-off. Instead of brute-forcing 2DES in 2^{112} steps, we can attack it in 2^{57} steps while storing 2^{56} values of the size of m.

Comment. This applies to any block cipher, not just DES!

Comment. For some block ciphers it is the case that for any pair of keys k_1, k_2, there is a third key k_3 such that $E_{k_2}(E_{k_1}(m)) = E_{k_3}(m)$. In that case, we say that the cipher is a group, and double (or triple, or quadruple) encryption does not add any additional security! DES, however, is not a group.

Example 118. Explain why 3DES, used with three different keys, only has effective key size 112.

Solution. (fill in the details!) Instead of going through all k_1, k_2, k_3 to check whether

$$c = E_{k_3}(D_{k_2}(E_{k_1}(m)))$$

(which would take $2^{56} \cdot 2^{56} \cdot 2^{56} = 2^{168}$ DES computations), we can use that the latter is equivalent to

$$D_{k_3}(c) = D_{k_2}(E_{k_1}(m)).$$

Now proceed as in the previous example ... to see that we can break 3DES with 2^{112} DES computations. How much memory do we need?

Example 119. (extra; use as PRG) ANSI X9.17 is a U.S. federal standard for a PRG based on 3DES.

Input: random, secret 64 bit seed s, key k for 3DES (keying option 2)

Produce a random number as follows:

- obtain current time D, compute $t = 3\mathrm{DES}_k(D)$
- output $x = 3\mathrm{DES}_k(s \oplus t)$ (that's the pseudo-random output)
- update the seed to $s = 3\mathrm{DES}_k(x \oplus t)$ for future use

Comment. ANSI (American National Standards Institute) X9 are standards for the financial industry.

https://en.wikipedia.org/wiki/Cryptographically_secure_pseudorandom_number_generator

Comment. The same approach can be applied to any block cipher.

Comment. It is common practice to add in time for PRGs that are used to generate enormous amounts of data. If nothing else, it slightly increases the entropy and reduces the likelihood of "short" periods.

Example 120. (extra; DES-X) To increase the key size of DES, the following variation, known as DES-X, was proposed by Ron Rivest in 1984:

$$c = k_3 \oplus \mathrm{DES}_{k_2}(m \oplus k_1)$$

What is the key size of DES-X? What about the effective key size?

Solution. k_1 and k_3 are 64 bit, while k_2 is 56 bits. That's a total key size of 184 bits for DES-X.

However, just like for 3DES, proceeding as in a meet-in-the-middle-attack (without the need of much storage) reduces the effective key size to at most $184 - 64 = 120$ bits.

Comment. This approach of xoring with a subkey before and after everything else is known as **key whitening**. This features in many modern ciphers, including AES.

https://en.wikipedia.org/wiki/DES-X

Block cipher modes

Block ciphers encrypt blocks of a specified size (64 bit for DES, or 128 bit for AES). **Block cipher modes** specify how to encrypt larger plaintexts.

Let E_k be the encryption routine of a block cipher with block size n bit. As a first step, we split a plaintext m into blocks $m = m_1 m_2 m_3 \ldots$ such that each m_i is n bits (we may have to pad).

Example 121. (ECB, shouldn't be used) In the simplest mode, known as **electronic codebook**, we just encrypt each plaintext block individually:

$$c_j = E_k(m_j)$$

The ciphertext is $c = c_1 c_2 c_3 \ldots$. Decryption simply computes $D_k(c_j) = m_j$.

Though natural, ECB has several severe weaknesses. Can you think of some?

Solution. Using ECB is nothing else but a classical substitution cipher, except that ECB operates on larger blocks. Just like a classical substitution cipher is vulnerable to frequency attacks, ECB leaves patterns in the ciphertext. For a striking visual example when encrypting a picture, see:

https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

If a block repeats later in the message (or in a later message), it will be encrypted the same way. Hence, Eve can notice such repetitions. This is problematic in practice, for instance, because certain files always begin with the same blocks, so that Eve has a good chance of detecting the file type.

Also, knowing the filetype, Eve might be able to rearrange the ciphertext blocks to adjust the message. She can also attempt to delete certain ciphertext blocks.

Conclusion. Unless you know exactly why (e.g. sending already randomized messages), you should not use ECB.

Example 122. (CBC) In **cipherblock chaining** mode, we encrypt each plaintext block after chaining it with the previous cipherblock; that is:

$$c_j = E_k(m_j \oplus c_{j-1})$$

In order to do that for $j = 1$, we need a value for c_0, known as an **initialization vector** IV.

The ciphertext is $c = c_0 c_1 c_2 c_3 \ldots$ (that's one more block than for the plaintext $m = m_1 m_2 m_3 \ldots$).

(a) How does decryption work?

(b) Why should the value IV be unpredictable (e.g. be chosen randomly)?

Solution.

(a) Since $c_j = E_k(m_j \oplus c_{j-1})$, we have $D_k(c_j) = m_j \oplus c_{j-1}$ or $m_j = D_k(c_j) \oplus c_{j-1}$.
For instance. $m_1 = D_k(c_1) \oplus c_0$

(b) The value IV should be unique, so that messages starting with the same plaintext block have different ciphertext blocks. More generally, it should be unpredictable so that Eve cannot mount a chosen-plaintext attack to test if an earlier plaintext equals her guess. See Example 124.

Just checking. What would happen if we set $c_j = E_k(m_j) \oplus c_{j-1}$ instead? In that case, we would gain nothing over ECB: since Eve knows all c_j, she can compute $c_j \oplus c_{j-1} = E_k(m_j)$.

Comment. CBC makes random access possible during decryption (but not encryption). That means, we don't need to decrypt $c = c_0 c_1 c_2 c_3 \ldots$ sequentially but can directly decrypt $c_N c_{N+1} \ldots$ for some random N.

Example 123. Consider the (silly) block cipher with 4 bit block size and 4 bit key size such that

$$E_k(b_1b_2b_3b_4) = (b_2b_3b_4b_1) \oplus k.$$

(a) Encrypt $m = (0000\ 1011\ 0000\ \ldots)_2$ using $k = (1111)_2$ and ECB mode.

(b) Encrypt $m = (0000\ 1011\ 0000\ \ldots)_2$ using $k = (1111)_2$ and CBC mode ($\mathrm{IV} = (0011)_2$).

Solution. $m = m_1m_2m_3\ldots$ with $m_1 = 0000$, $m_2 = 1011$ and $m_3 = 0000$.

(a) $c_1 = E_k(m_1) = 0000 \oplus 1111 = 1111$
$c_2 = E_k(m_2) = 0111 \oplus 1111 = 1000$
Since $m_3 = m_1$, we have $c_3 = c_1$. Hence, the ciphertext is $c = c_1c_2c_3\ldots = (1111\ 1000\ 1111\ \ldots)$.

(b) $c_0 = 0011$
$c_1 = E_k(m_1 \oplus c_0) = E_k(0000 \oplus 0011) = E_k(0011) = 0110 \oplus 1111 = 1001$
$c_2 = E_k(m_2 \oplus c_1) = E_k(1011 \oplus 1001) = E_k(0010) = 0100 \oplus 1111 = 1011$
$c_3 = E_k(m_3 \oplus c_2) = E_k(0000 \oplus 1011) = E_k(1011) = 0111 \oplus 1111 = 1000$
Hence, the ciphertext is $c = c_0c_1c_2c_3\ldots = (0011\ 1001\ 1011\ 1000\ \ldots)$.

Comment. Clearly, our cipher is not meant to be secure. One damning issue (besides the short key and block size) is that it is linear (in both the plaintext and the key).

Extra. In each case, can you decrypt c to get back the original m?

Example 124. (BEAST attack) BEAST is short for Browser Exploit Against SSL/TLS and was brought to public attention in 2011. The attack is based on the fact that the IV used by SSL was obtained from a previous ciphertext block (instead of randomly!):

Scenario. Imagine that plaintext blocks $m_1m_2\ldots$ are continuously being encrypted using CBC to cipherblocks. However, the plaintexts are from different parties and Eve can ask for her own plaintexts to be encrypted along the way.

In such a scenario, different plaintexts should be separately encrypted using CBC, meaning that a new random IV should be chosen each time.

Eve's goal. Suppose Eve has observed the ciphertext blocks c_{j-1}, c_j and her goal is to find out whether $m_j = x$ where x is her educated guess. Obviously, this is something that Eve should not be able to do!

The exploit. Because the IV for the next encryption is c_j, and because Eve can interject plaintext blocks to be encrypted for her, she can ask for $m_{j+1} = x \oplus c_{j-1} \oplus c_j$ (these are all known to Eve!) to be encrypted next.

Because CBC with IV c_j is used, this results in $c_{j+1} = E_k(m_{j+1} \oplus c_j) = E_k(x \oplus c_{j-1})$.

Eve can now compare this with $E_k(m_j \oplus c_{j-1}) = c_j$ (which she knows!) to find out whether $m_j = x$.

https://en.wikipedia.org/wiki/Transport_Layer_Security#BEAST_attack

There exist many other modes, including modes which already include features like authentication. Other common basic modes such as OFB (output feedback) or CTR (counter) turn the block cipher into a stream cipher (one advantage of that is that we don't need to encrypt full blocks at a time).

https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

Comment. One issue of ECB and CBC is the need for padding. If not handled properly, this can be exploited by a **padding oracle attack**:

https://en.wikipedia.org/wiki/Padding_oracle_attack

Example 125. (bonus challenge!) Find the smallest (pseudo)prime with 100 decimal digits, all of which are 1 or 2.

(Send me an email by 3/15 with the prime, and how you found it, to collect a bonus point. Earn an extra bonus point if you can find it using a single line of Sage code [artificial concatenations not allowed].)

AES

Finite fields

Example 126. We have already seen xor in several cryptosystems. Note that a single xor operation as in the one-time pad or stream ciphers provides no diffusion.

When designing a cipher it may be nice to replace xor of N bit blocks with an operation that does provide some diffusion.

- A tiny amount of diffusion is provided by instead using addition modulo 2^N.
  Due to carries, one bit flip in the input can propagate to more than one bit flipped in the output.
- More diffusion can be achieved using operations (multiplication/inversion) in finite fields like $\mathrm{GF}(2^N)$.
  [We only need to make sure in our design that we don't multiply with zero.]

A **field** is a set of elements which can be added/subtracted as well as multiplied/divided by according to the usual rules.

In particular, a field always has distinguished elements 0 and 1, which are the neutral elements with respect to addition and multiplication, respectively.

Example 127.

- The rational numbers $\mathbb{Q}$, the real numbers $\mathbb{R}$, and the complex numbers $\mathbb{C}$ all are fields, which you have seen before. They contain infinitely many elements.
- The integers $\mathbb{Z}$ are not a field because, for instance, 3 is not invertible (since $\frac{1}{3}$ is not an integer itself). Quotients of integers (rational numbers!) are a field.
  Since addition/subtraction and multiplication work as they should, $\mathbb{Z}$ is what is called a **ring**.
- Polynomials are not a field (they are a ring like $\mathbb{Z}$). Quotients of polynomials (rational functions!) are a field.

Cryptographic applications require finite structures. Correspondingly, our focus will be on **finite fields**, that is, fields consisting of only a finite number of elements.

Example 128. Let p be a prime. The residues modulo p form a field, often denoted as $\mathrm{GF}(p)$.

GF is short for **Galois field**, which is another word for finite field.

Note that we can divide by any element! (Except the zero residue but, of course, we can never divide by 0).

Example 129. The residues modulo 21 (or any other composite number) are not a field.

We can add/subtract and multiply these numbers, but we cannot always divide. Specifically, we cannot divide by elements like $3, 6, 7, \ldots$ even though these are nonzero (we can, of course, never divide by zero).

Note. We have already seen that this seemingly slight deficiency has "terrible" consequences. For instance, the quadratic equation $x^2 = 1$ has more than the two solutions $x = \pm 1$ modulo 21 (namely, ± 8 as well).

AES is built upon byte operations (in contrast to DES, which is built on bit operations). Each of the 2^8 bytes represents one of the 2^8 elements of the finite field $\mathrm{GF}(2^8)$.

Note. We do not yet know what $\mathrm{GF}(2^8)$ is. It cannot be the residues modulo 2^8, because we just observed that the residues modulo n are a field only if n is prime.

Armin Straub
straub@southalabama.edu

To construct the finite field $\mathrm{GF}(p^n)$ of p^n elements, we can do the following:

- Fix a polynomial $m(x)$ of degree n, which is irreducible modulo p (i.e. cannot be factored modulo p).
- The elements of $\mathrm{GF}(p^n)$ are polynomials modulo $m(x)$ modulo p.

We will discuss the irreducibility condition on $m(x)$ next time. For now, see Example 132.

Comment. Actually, all finite fields can be constructed in this fashion. Moreover, choosing different $m(x)$ to construct $\mathrm{GF}(p^n)$ does not really matter: the resulting fields are always isomorphic (i.e. work in the same way, although the elements are represented differently). That justifies writing down $\mathrm{GF}(p^n)$, since there is exactly one such field.

Example 130. AES is based on representing bytes as elements of the field $\mathrm{GF}(2^8)$. It is constructed using the polynomial $x^8+x^4+x^3+x+1$ (which is indeed irreducible mod 2).

From bits to polynomials. For instance, the polynomial x^7+x^4+x corresponds to the bits 10010010 while x^6+1 corresponds to 01000001.

Example 131. The polynomial x^2+x+1 is irreducible modulo 2, so we can use it to construct the finite field $\mathrm{GF}(2^2)$ with 4 elements.

(a) List all 4 elements, and make an addition table. Then realize that this is just xor.

(b) Make a multiplication table.

(c) What is the inverse of $x+1$?

Solution.

(a) The four elements are $0, 1, x, x+1$.

For instance, $(x+1)+x=2x+1=1$ (in $\mathrm{GF}(2^2)$, since we are working modulo 2). The full table is below.

Each of the four elements is of the form $ax+b$, which can be represented using the two bits ab (for instance, $(10)_2$ represents x and $(11)_2$ represents $x+1$).

Then, addition of elements $ax+b$ in $\mathrm{GF}(2^2)$ works in the same way as xoring bits ab.

(b) For instance, $(x+1)^2=x^2+2x+1\equiv x^2+1\equiv(x+1)+1\equiv x$.

Here, the key is to realize that reducing modulo x^2+x+1 is the same as saying that $x^2=-x-1$, i.e. $x^2=x+1$ in $\mathrm{GF}(2^2)$. That means all polynomials of degree 2 and higher can be reduced to polynomials of degree less than 2.

$+$	0	1	x	$x+1$
0	0	1	x	$x+1$
1	1	0	$x+1$	x
x	x	$x+1$	0	1
$x+1$	$x+1$	x	1	0

$\times$	0	1	x	$x+1$
0	0	0	0	0
1	0	1	x	$x+1$
x	0	x	$x+1$	1
$x+1$	0	$x+1$	1	x

(c) We are looking for an element y such that $y(x+1)=1$ in $\mathrm{GF}(2^2)$. Looking at the table, we see that $y=x$ has that property. Hence, $(x+1)^{-1}=x$ in $\mathrm{GF}(2^2)$.

Example 132. What if we proceed as in the previous example but used $m(x)=x^2+1$ instead?

Solution. The addition table would be the same. The multiplication table would be different and a crucial difference would be that $(x+1)\cdot(x+1)=x^2+2x+1\equiv x^2+1\equiv 0$, which implies that $x+1$ cannot be invertible. That means our construction is not a field.

Comment. Note how, here, $m(x)$ factors modulo 2 as $x^2+1\equiv(x+1)(x+1)$. Hence the condition of irreducibility in the construction of $\mathrm{GF}(p^n)$ is violated.

Review. $\mathrm{GF}(p^n)$ is "the" finite field with p^n elements.

Recall that, in the construction of $\mathrm{GF}(p^n)$, the polynomial $m(x)$ has to be such that it cannot be factored modulo p. We also say that $m(x)$ needs to be **irreducible** mod p.

For instance. The polynomial x^2+2x+1 can always be factored as $(x+1)^2$.

On the other hand. For the polynomials $m(x)=x^2+x+1$ things are more interesting:

- x^2+x+1 cannot be factored over $\mathbb{Q}$ because the roots $\frac{-1\pm\sqrt{-3}}{2}$ are not rational.
- However, $x^2+x+1\equiv(x+2)^2$ modulo 3, so it can be factored modulo 3.
- On the other hand, x^2+x+1 is irreducible modulo 2 (that is, it cannot be factored: the only linear factors are x and $x+1$, but x^2, $x(x+1)$ and $(x+1)^2$ are all different from x^2+x+1 modulo 2).

In general, it follows from the formula $\frac{-1\pm\sqrt{-3}}{2}$ for the roots that x^2+x+1 can be factored modulo a prime $p>2$ if and only if $\sqrt{-3}$ exists as a residue modulo p. In other words, if and only if -3 is a quadratic residue modulo p.

For instance. Modulo $p=7$, we have $-3\equiv 2^2$ and $\frac{1}{2}\equiv 4$, so that $\frac{-1\pm\sqrt{-3}}{2}\equiv 4\cdot(-1\pm 2)\equiv 2,4$. Indeed, we have the factorization $(x-2)(x-4)=x^2-6x+8\equiv x^2+x+1$ modulo 7.

Example 133. The polynomial x^3+x+1 is irreducible modulo 2, so we can use it to construct the finite field $\mathrm{GF}(2^3)$ with 8 elements.

(a) List all 8 elements.

(b) Reduce x^5+1 in $\mathrm{GF}(2^3)$.

(c) Multiply each element of $\mathrm{GF}(2^3)$ with x^2+x.

(d) What is the inverse of x^2+x in $\mathrm{GF}(2^3)$?

Solution.

(a) The elements are $0,1,x,x+1,x^2,x^2+1,x^2+x,x^2+x+1$.
[Note that $x^3=-x-1=x+1$ in $\mathrm{GF}(2^3)$. That means all polynomials of degree 3 and higher can be reduced to polynomials of degree less than 3. See next part.]

(b) We divide x^5+1 by x^3+x+1 (long division!) to find $x^5+1=(x^2-1)(x^3+x+1)+(-x^2+x+2)$.
It follows that x^5+1 reduces to $-x^2+x+2\equiv x^2+x$ in $\mathrm{GF}(2^3)$.
Important. We can simplify things by performing the long division modulo 2. We then find $x^5+1\equiv(x^2+1)(x^3+x+1)+(x^2+x)$.

(c) We multiply the polynomials as usual, then reduce as in the previous part.
For instance, $(x^2+x)(x^2+x+1)\equiv x^4+x$ and, by long division, $x^4+x\equiv x(x^3+x+1)+x^2$, which reduces to just x^2 in $\mathrm{GF}(2^3)$.

$\times$	0	1	x	$x+1$	x^2	x^2+1	x^2+x	x^2+x+1
x^2+x	0	x^2+x	x^2+x+1	1	x^2+1	$x+1$	x	x^2

(d) We are looking for an element y such that $y(x^2+x)=1$ in $\mathrm{GF}(2^3)$. Looking at the table, we see that $y=x+1$ has that property. Hence, $(x^2+x)^{-1}=x+1$ in $\mathrm{GF}(2^3)$.
Important. To find the inverse, we essentially tried all possibilities. That's not sustainable. Instead, we can (and should!) proceed as we did for computing the inverse of residues modulo n. That is, we should use the Euclidean algorithm as indicated in the next examples. Here, this is just one step: modulo 2, we have $\boxed{x^3+x+1}\equiv(x+1)\cdot\boxed{x^2+x}+1$, so that $(x^2+x)^{-1}=x+1$ in $\mathrm{GF}(2^3)$.

Example 134.

(a) Apply the extended Euclidean algorithm to find the gcd of x^2+1 and x^4+x+1, and spell out Bezout's identity.

(b) Repeat the previous computation but always reduce all coefficients modulo 2.

(c) What is the inverse of x^2+1 in $\mathrm{GF}(2^4)$? Here, $\mathrm{GF}(2^4)$ is constructed using x^4+x+1.

Solution.

(a) We use the extended Euclidean algorithm:

$$\begin{aligned}&\gcd(x^2+1, x^4+x+1) && \boxed{x^4+x+1} = (x^2-1)\cdot\boxed{x^2+1} + (x+2)\\ &= \gcd(x+2, x^2+1) && \boxed{x^2+1} = (x-2)\cdot\boxed{x+2} + 5\\ &= 5\end{aligned}$$

Backtracking through this, we find that Bézout's identity takes the form

$$\begin{aligned}5 \;=\; 1\cdot\boxed{x^2+1} - (x-2)\cdot\boxed{x+2} \;&=\; 1\cdot\boxed{x^2+1} - (x-2)\cdot\left(\boxed{x^4+x+1} - (x^2-1)\cdot\boxed{x^2+1}\right)\\ &=\; (x^3-2x^2-x+3)\cdot\boxed{x^2+1} - (x-2)\cdot\boxed{x^4+x+1}\end{aligned}$$

If we wanted to, we could divide both sides by 5.

(b) We repeat the exact same computation but reduce modulo 2 at each step:

$$\begin{aligned}\boxed{x^4+x+1} \;&\equiv\; (x^2+1)\cdot\boxed{x^2+1} + x\\ \boxed{x^2+1} \;&\equiv\; = x\cdot\boxed{x} + 1\end{aligned}$$

Backtracking through this, we find that Bézout's identity takes the form

$$\begin{aligned}1 \;=\; 1\cdot\boxed{x^2+1} + x\cdot\boxed{x} \;&=\; 1\cdot\boxed{x^2+1} + x\cdot\left(\boxed{x^4+x+1} + (x^2+1)\cdot\boxed{x^2+1}\right)\\ &=\; (x^3+x+1)\cdot\boxed{x^2+1} + x\cdot\boxed{x^4+x+1}\end{aligned}$$

(c) We can now read off that $(x^2+1)^{-1} = x^3+x+1$ in $\mathrm{GF}(2^4)$.

Example 135. (HW) Find the inverses of x^2+1 and x^3+1 in $\mathrm{GF}(2^8)$, constructed as in AES.

Solution. Recall that for AES, $\mathrm{GF}(2^8)$ is constructed using $x^8+x^4+x^3+x+1$.

(a) We use the extended Euclidean algorithm for polynomials, and reduce all coefficients modulo 2:

$$\boxed{x^8+x^4+x^3+x+1} \equiv (x^6+x^4+x)\cdot\boxed{x^2+1} + 1$$

Hence, $(x^2+1)^{-1} = x^6+x^4+x$ in $\mathrm{GF}(2^8)$.

(b) We use the extended Euclidean algorithm, and always reduce modulo 2:

$$\begin{aligned}\boxed{x^8+x^4+x^3+x+1} \;&\equiv\; (x^5+x^2+x+1)\cdot\boxed{x^3+1} + x^2\\ \boxed{x^3+1} \;&\equiv\; x\cdot\boxed{x^2} + 1\end{aligned}$$

Backtracking through this, we find that Bézout's identity takes the form

$$\begin{aligned}1 \;\equiv\; 1\cdot\boxed{x^3+1} - x\cdot\boxed{x^2} \;&\equiv\; 1\cdot\boxed{x^3+1} - x\cdot\left(\boxed{x^8+x^4+x^3+x+1} - (x^5+x^2+x+1)\cdot\boxed{x^3+1}\right)\\ &\equiv\; (x^6+x^3+x^2+x+1)\cdot\boxed{x^3+1} + x\cdot\boxed{x^8+x^4+x^3+x+1}.\end{aligned}$$

Hence, $(x^3+1)^{-1} = x^6+x^3+x^2+x+1$ in $\mathrm{GF}(2^8)$.

Armin Straub
straub@southalabama.edu

Basics of AES

The block cipher AES (short for **advanced encryption standard**) replaced DES. By now, it is the most important symmetric block cipher.

1997: NIST requests proposals for AES (receives 15 submissions) [very different from how DES was selected!]

2000: Rijndael (by Joan Daemen and Vincent Rijmen) selected (from 5 finalists)

https://csrc.nist.gov/csrc/media/publications/fips/197/final/documents/fips-197.pdf

- 128 bit block size (as per NIST request)
- The key size of AES can be 128, 192 or 256 bit. The corresponding choices are referred to as AES-128, AES-192 and AES-256, and have 10, 12 and 14 rounds, respectively.
- AES-192/256 is first (and only) public cipher allowed by NSA for top secret information.
- No known attacks on AES which are substantially better than brute-force.

Attacks better than brute-force known if the number of rounds was 6 (instead of 10) for AES-128.

- Unlike DES, AES is not a Feistel network.

While for a Feistel network, each round only encrypts half of the bits, all bits are being encrypted during each round. That's one indication why AES requires less rounds than DES.

Internals of AES

Each round consists of 4 **layers**. Each layer takes 128 bits input and outputs 128 bits in a reversible way (so that we can decrypt as long as we know the key).The 128 bit state consists of 16 bytes. These 16 bytes $c_{0,0}, c_{1,0}, c_{2,0}, c_{3,0}, c_{0,1}, \dots, c_{3,3}$ are often arranged in a 4x4 matrix as

$$\begin{bmatrix} c_{0,0} & c_{0,1} & c_{0,2} & c_{0,3} \\ c_{1,0} & c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,0} & c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,0} & c_{3,1} & c_{3,2} & c_{3,3} \end{bmatrix}.$$

Each byte is identified with an element of $\mathrm{GF}(2^8)$.

Example 136. $(0000\ 0101)_2$ represents the element x^2+1 in $\mathrm{GF}(2^8)$.

The 4 layers are:

- ByteSub

  each byte gets substituted with another byte (like a single S-box in DES); provides confusion and guarantees non-linearity of AES

- ShiftRow

  the 16 bytes are permuted (like a P-box in DES but on bytes, not bits); provides diffusion

- MixCol

  the 4x4 matrix is linearly transformed; provides diffusion

- AddRoundKey

  the state is xored with a 128 bit round key

Slight deviations. Before the first round, AddRoundKey is applied with the 0th round key (which equals the AES key). Otherwise, our first step would be ByteSub, which wouldn't have any cryptographic effect since the plaintext bytes would just be changed in a fixed manner (no key involved yet).

Also, the last round has no MixCol layer. This has the effect that decryption can be made to look very much like encryption (see Section 5.3 in our book for the details).

ShiftRow

The layer **ShiftRow** permutes the 16 bytes $c_{0,0}, c_{1,0}, c_{2,0}, c_{3,0}, c_{0,1}, \ldots, c_{3,3}$ as follows:

$$\begin{bmatrix} c_{0,0} & c_{0,1} & c_{0,2} & c_{0,3} \\ c_{1,0} & c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,0} & c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,0} & c_{3,1} & c_{3,2} & c_{3,3} \end{bmatrix} \mapsto \begin{bmatrix} c_{0,0} & c_{0,1} & c_{0,2} & c_{0,3} \\ c_{1,1} & c_{1,2} & c_{1,3} & c_{1,0} \\ c_{2,2} & c_{2,3} & c_{2,0} & c_{2,1} \\ c_{3,3} & c_{3,0} & c_{3,1} & c_{3,2} \end{bmatrix}$$

MixCol

Again, arrange the 16 bytes as a 4×4 matrix with entries in $\mathrm{GF}(2^8)$. The **MixCol** layer transform this 4×4 matrix by multiplying it with another, fixed, 4×4 matrix:

$$\begin{bmatrix} c_{0,0} & c_{0,1} & c_{0,2} & c_{0,3} \\ c_{1,0} & c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,0} & c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,0} & c_{3,1} & c_{3,2} & c_{3,3} \end{bmatrix} \mapsto \begin{bmatrix} x & x+1 & 1 & 1 \\ 1 & x & x+1 & 1 \\ 1 & 1 & x & x+1 \\ x+1 & 1 & 1 & x \end{bmatrix} \begin{bmatrix} c_{0,0} & c_{0,1} & c_{0,2} & c_{0,3} \\ c_{1,0} & c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,0} & c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,0} & c_{3,1} & c_{3,2} & c_{3,3} \end{bmatrix}$$

Example 137. For instance, the new byte at the position $c_{2,1}$ is

$$\begin{bmatrix} 1 & 1 & x & x+1 \end{bmatrix} \begin{bmatrix} c_{0,1} \\ c_{1,1} \\ c_{2,1} \\ c_{3,1} \end{bmatrix} = c_{0,1} + c_{1,1} + x\,c_{2,1} + (x+1)c_{3,1},$$

where all computations are to be done in $\mathrm{GF}(2^8)$.

AddRoundKey

The **AddRoundKey** layer simply xors the current 128 bit state with a 128 bit round key.

The **key schedule** for AES-128 is as follows. Like for the states, arrange the original 16 byte AES key k in a 4×4 matrix with columns $W(0), W(1), W(2), W(3)$.

The ith round key is then obtained from the matrix with columns $W(4i), W(4i+1), W(4i+2), W(4i+3)$, where $W(4), W(5), \ldots$ are recursively constructed:

$$W(i) = \begin{cases} W(i-4) + W(i-1), & \text{if } 4 \nmid i, \\ W(i-4) + \tilde{W}(i-1), & \text{if } 4 | i. \end{cases}$$

Here, $\tilde{W}(i-1)$ is obtained from $W(i-1)$ as follows:

$$W(i-1) = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} \quad \Longrightarrow \quad \tilde{W}(i-1) = \begin{bmatrix} S(w_1) + x^{(i-4)/4} \\ S(w_2) \\ S(w_3) \\ S(w_4) \end{bmatrix}.$$

Note that w_1, w_2, w_3, w_4 each are bytes. The function S is the ByteSub substitution. Since that substitution is nonlinear, the round keys are constructed from k in a nonlinear manner (unlike in DES).

As usual, the computation $S(w_1) + x^{(i-4)/4}$ happens in $\mathrm{GF}(2^8)$.

ByteSub

The **ByteSub** layer takes each of the 16 bytes y and replaces it with the byte $S(y)$. As in DES, we could simply describe the (invertible) map by a lookup table. However, like the other steps of AES, it has a very simple mathematical description which we'll discuss next time.

Comment. As for the S-boxes in DES, ByteSub can be implemented in hardware as a lookup table. Since we have $2^8 = 256$ inputs, with 1 byte of output each, this table is 256 bytes large. (See Table 5.1 in our book.)

For comparison, each of the eight S-boxes in DES occupies 2^6 times 4 bits, which is 32 bytes. In total, these are also 256 bytes.

[In contrast to DES it is the case (and necessary for decryption!) that different inputs have different outputs.]

- Recall that, in contrast to DES, the operations of AES have very simple (though somewhat advanced) mathematical descriptions.

  No mysteriously constructed S-boxes and P-boxes as in DES.

ByteSub (continued)

Each of the 16 bytes gets substituted as follows.

Note. The mathematical description below can be implemented in a **lookup table**: you can find this table in Table 5.1 of our book or, for instance, on wikipedia: https://en.wikipedia.org/wiki/Rijndael_S-box

- Interpret the input byte $(b_7b_6...b_0)_2$ as the element $b_7x^7+...+b_1x+b_0$ of $\mathrm{GF}(2^8)$.
- Compute $s^{-1}=c_0+c_1x+...+c_7x^7$ (with 0^{-1} interpreted as 0).

  Important comment. This inversion is what makes AES highly nonlinear.

  If the ByteSub substitution was linear, then all of AES would be linear (because all other layers are linear; assuming we adjust the key schedule accordingly).

- Then the output bits $(d_7d_6...d_1d_0)_2$ are

$$\begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ d_7 \end{bmatrix} = \begin{bmatrix} 1&0&0&0&1&1&1&1 \\ 1&1&0&0&0&1&1&1 \\ 1&1&1&0&0&0&1&1 \\ 1&1&1&1&0&0&0&1 \\ 1&1&1&1&1&0&0&0 \\ 0&1&1&1&1&1&0&0 \\ 0&0&1&1&1&1&1&0 \\ 0&0&0&1&1&1&1&1 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}.$$

Comment. The particular choice of matrix and vector has the effect that no ByteSub output equals the ByteSub input (or its complement).

Example 138. Invert x^3+1 in $\mathrm{GF}(2^8)$, constructed as in AES. [Example 135, again]

Solution. We use the extended Euclidean algorithm, and always reduce modulo 2:

$$\begin{aligned} \boxed{x^8+x^4+x^3+x+1} &\equiv (x^5+x^2+x+1)\cdot\boxed{x^3+1}+x^2 \\ \boxed{x^3+1} &\equiv x\cdot\boxed{x^2}+1 \end{aligned}$$

Backtracking through this, we find that Bézout's identity takes the form

$$\begin{aligned} 1 \equiv 1\cdot\boxed{x^3+1}-x\cdot\boxed{x^2} &\equiv 1\cdot\boxed{x^3+1}-x\cdot\left(\boxed{x^8+x^4+x^3+x+1}-(x^5+x^2+x+1)\cdot\boxed{x^3+1}\right) \\ &\equiv (x^6+x^3+x^2+x+1)\cdot\boxed{x^3+1}+x\cdot\boxed{x^8+x^4+x^3+x+1}. \end{aligned}$$

Hence, $(x^3+1)^{-1}=x^6+x^3+x^2+x+1$ in $\mathrm{GF}(2^8)$.

Example 139. (homework)

(a) What happens to the byte $(0000\ 0101)_2$ during ByteSub?

(b) What happens to the byte $(0000\ 1001)_2$ during ByteSub?

Armin Straub
straub@southalabama.edu

Solution.

(a) $(0000\ 0101)_2$ represents the polynomial x^2+1.
By the previous example, its inverse is $(x^2+1)^{-1} = x^6+x^4+x$ in $\mathrm{GF}(2^8)$, which is $\boldsymbol{c} = (0101\ 0010)_2$.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \underbrace{\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}}_{\boldsymbol{c}} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

[This is just the usual matrix-vector product modulo 2. The highlighted columns are the ones which get added up during this matrix-vector product.]

Hence, the output of ByteSub is the byte $(0110\ 1011)_2$.

Check with lookup tables. Indeed, our computation matches $107 = (0110\ 1011)_2$ in the lookup table in our book (row 0, column $(0101)_2 = 5$) or $(6\mathrm{B})_{16} = (0110\ 1011)_2$ on wikipedia (row $(0000)_2 = (0)_{16}$, column $(0101)_2 = (5)_{16}$).

(b) $(0000\ 1001)_2$ represents the polynomial x^3+1.
By the previous example, $(x^3+1)^{-1} = x^6+x^3+x^2+x+1$ in $\mathrm{GF}(2^8)$, which is $\boldsymbol{c} = (0100\ 1111)_2$.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \underbrace{\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}}_{\boldsymbol{c}} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Hence, the output of ByteSub is the byte $(0000\ 0001)_2$.

Check with lookup tables. Indeed, our computation matches the value 1 in the lookup table in our book (row 0, column $(1001)_2 = 9$) or $(01)_{16}$ on wikipedia (row $(0000)_2 = (0)_{16}$, column $(1001)_2 = (9)_{16}$).

Review: multiplicative order and primitive roots

Definition 140. The **multiplicative order** of an invertible residue a modulo n is the smallest positive integer k such that $a^k \equiv 1 \pmod{n}$.

Important note. By Euler's theorem, the multiplicative order can be at most $\phi(n)$.

Example 141. What is the multiplicative order of $2 \pmod 7$?

Solution. $2^1 = 2$, $2^2 = 4$, $2^3 \equiv 1 \pmod 7$. Hence, the multiplicative order of $2 \pmod 7$ is 3.

Definition 142. If the multiplicative order of an residue a modulo n equals $\phi(n)$ [in other words, the order is as large as possible], then a is said to be **primitive root** modulo n.

A primitive root is also referred to as a **multiplicative generator** (because the products of a and itself, that is, $1, a, a^2, a^3, \ldots$, produce all invertible residues).

Example 143. What is the multiplicative order of $3 \pmod 7$?

Solution. $3^1 = 3$, $3^2 \equiv 2$, $3^3 \equiv 6$, $3^4 \equiv 4$, $3^5 \equiv 5$, $3^6 \equiv 1$. Hence, the multiplicative order of $3 \pmod 7$ is 6. This means that 3 is a primitive root modulo 7. Note how every (invertible) residue shows up as a power of 3.

Review. $x \pmod n$ is a primitive root.

$\iff$ The (multiplicative) order of $x \pmod n$ is $\phi(n)$. (That is, the order is as large as possible.)

$\iff$ $x, x^2, \ldots, x^{\phi(n)}$ is a list of all invertible residues modulo n.

Lemma 144. If $a^r \equiv 1 \pmod n$ and $a^s \equiv 1 \pmod n$, then $a^{\gcd(r,s)} \equiv 1 \pmod n$.

Proof. By Bezout's identity, there are integers x, y such that $xr + ys = \gcd(r,s)$.
Hence, $a^{\gcd(r,s)} = a^{xr+ys} = a^{xr}a^{ys} = (a^r)^x (a^s)^y \equiv 1 \pmod n$. □

Corollary 145. The multiplicative order of a modulo n divides $\phi(n)$.

Proof. Let k be the multiplicative order, so that $a^k \equiv 1 \pmod n$. By Euler's theorem $a^{\phi(n)} \equiv 1 \pmod n$. The previous lemma shows that $a^{\gcd(k,\phi(n))} \equiv 1 \pmod n$. But since the multiplicative order is the smallest exponent, it must be the case that $\gcd(k, \phi(n)) = k$. Equivalently, k divides $\phi(n)$. □

Comment. By the same argument, if $a^m \equiv 1 \pmod n$, then the order of $a \pmod n$ divides m.

Example 146. Compute the multiplicative order of 2 modulo $7, 11, 9, 15$. In each case, is 2 a primitive root?

Solution.

- $2 \pmod 7$: $2^2 \equiv 4, 2^3 \equiv 1$. Hence, the order of 2 modulo 7 is 3.
  Since the order is less than $\phi(7) = 6$, 2 is not a primitive root modulo 7.
- $2 \pmod{11}$: Since $\phi(11) = 10$, the only possible orders are $2, 5, 10$. Hence, checking that $2^2 \not\equiv 1$ and $2^5 \not\equiv 1$ is enough to conclude that the order must be 10.
  Since the order is equal to $\phi(11) = 10$, 2 is a primitive root modulo 11.
  Brute force approach (too much unnecessary work). Just for comparison, $2^0 = 1, 2^1 = 2, 2^2 = 4, 2^3 = 8, 2^4 \equiv 5, 2^5 \equiv 2 \cdot 5 = 10, 2^6 \equiv 2 \cdot 10 \equiv 9, 2^7 \equiv 2 \cdot 9 \equiv 7, 2^8 \equiv 2 \cdot 7 \equiv 3, 2^9 \equiv 2 \cdot 3 = 6, 2^{10} \equiv 2 \cdot 6 \equiv 1$. Thus, the order of 2 mod 11 is 10.
- $2 \pmod 9$: Since $\phi(9) = 6$, the only possible orders are $2, 3, 6$. Hence, checking that $2^2 \not\equiv 1$ and $2^3 \not\equiv 1$ is enough to conclude that the order must be 6. (Indeed, $2^2 \equiv 4$, $2^3 \equiv 8$, $2^4 \equiv 7$, $2^5 \equiv 5$, $2^6 \equiv 1$.)
  Since the order is equal to $\phi(9) = 6$, 2 is a primitive root modulo 9.
- The order of $2 \pmod{15}$ is 4 (a divisor of $\phi(15) = 8$).
  2 is not a primitive root modulo 15. In fact, there is no primitive root modulo 15.

Comment. It is an open conjecture to show that 2 is a primitive root modulo infinitely many primes. (This is a special case of Artin's conjecture which predicts much more.)

Advanced comment. There exists a primitive root modulo n if and only if n is of one of $1, 2, 4, p^k, 2p^k$ for some odd prime p.

Example 147. Show that $x^4 \equiv 1 \pmod{15}$ for all invertible residues $x \pmod{15}$. In particular, there are no primitive roots modulo 15.

Solution. By the Chinese Remainder Theorem:

$x^4 \equiv 1 \pmod{15}$
$\iff$ $x^4 \equiv 1 \pmod 3$ and $x^4 \equiv 1 \pmod 5$

The congruences modulo 3 and 5 follow immediately from Fermat's little theorem.

Comment. The same argument shows that there are no primitive roots modulo pq, where p and q are distinct odd primes (because each element has order dividing $\phi(pq)/2$).

Armin Straub
straub@southalabama.edu

Lemma 148. Suppose $x \pmod n$ has (multiplicative) order k.

(a) $x^a \equiv 1 \pmod n$ if and only if $k|a$.

(b) x^a has order $\frac{k}{\gcd(k,a)}$.

Proof.

(a) "$\Longrightarrow$": By Lemma 144, $x^k \equiv 1$ and $x^a \equiv 1$ imply $x^{\gcd(k,a)} \equiv 1 \pmod n$. Since k is the smallest exponent, we have $k = \gcd(k,a)$ or, equivalently, $k|a$.
"$\Longleftarrow$": Obviously, if $k|a$ so that $a = kb$, then $x^a = (x^k)^b \equiv 1 \pmod n$.

(b) By the first part, $(x^a)^m \equiv 1 \pmod n$ if and only if $k|am$. The smallest such m is $m = \frac{k}{\gcd(k,a)}$. □

Example 149. Determine the orders of each (invertible) residue modulo 7. In particular, determine all primitive roots modulo 7.

Solution. First, observe that, since $\phi(7) = 6$, the orders can only be $1, 2, 3, 6$. Indeed:

residues	1	2	3	4	5	6
order	1	3	6	3	6	2

The primitive roots are 3 and 5.

Example 150. Redo Example 149, starting with the knowledge that 3 is a primitive root.

Solution.

residues	1	2	3	4	5	6
3^a	3^0	3^2	3^1	3^4	3^5	3^3
order$=\frac{6}{\gcd(a,6)}$	$\frac{6}{6}$	$\frac{6}{2}$	$\frac{6}{1}$	$\frac{6}{2}$	$\frac{6}{1}$	$\frac{6}{3}$

RSA and public key cryptography

- So far, our symmetric ciphers required a single **private key** k, a secret shared between the communicating parties.

  That leaves the difficult task of how to establish such private keys over a medium like the internet.

- In **public key cryptosystems**, there are two keys k_e, k_d, one for encryption and one for decryption. Bob keeps k_d secret (from anyone else!) and shares k_e with the world. Alice (or anyone else) can then send an encrypted message to Bob using k_e. However, Bob is the only who can decrypt it using k_d.

  It is crucial that the key k_d cannot be (easily) constructed from k_e.

RSA is one the first public key cryptosystems.

- It was described by Ron Rivest, Adi Shamir, and Leonard Adleman in 1977. (Note the initials!)
- However, a similar system had already been developed in 1973 by Clifford Cocks for the UK intelligence agency GCHQ (classified until 1997). Even earlier, in 1970, his colleague James Ellis was likely the first to discover public key cryptography.

Example 151. Let us emphasize that it should be surprising that something like public key cryptography is even possible.

Imagine Alice, Bob and Eve sitting at a table. Everything that is being said is heard by all three of them. The three have never met before and share no secrets. Should it be possible in these circumstances that Alice and Bob can share information without Eve also learning about it?

Public key cryptography makes exactly that possible!

Comments on primitive roots

Example 152. Determine all primitive roots modulo 11.

Solution. Since $\phi(11)=10$, the possible orders of residues modulo 11 are $1,2,5,10$. Residues with order 10 are primitive roots. Our strategy is to find one primitive root and to use that to compute all primitive roots.
There is no good way of finding the first primitive root. We will just try the residues $2,3,5,\ldots$ (why not 4?!)
We compute the order of $2 \pmod{11}$:
Since $2^2=4\not\equiv 1$, $2^5\equiv -1\not\equiv 1 \pmod{11}$, we find that 2 has order 10. Hence, 2 is a primitive root.
All other invertible residues are of the form 2^x. Recall that the order of $2^x \pmod{11}$ is $\frac{10}{\gcd(10,x)}$.
Hence, 2^x is a primitive root if and only if $\gcd(10,x)=1$, which yields $x=1,3,7,9$.
In conclusion, the primitive roots modulo 11 are $2^1=2, 2^3=8, 2^7\equiv 7, 2^9\equiv 6$.

Example 153. (extra) Determine all primitive roots modulo 22.

Solution. We proceed as in the previous example:

- Since $\phi(22)=10$, the possible orders of residues modulo 22 are $1,2,5,10$.
- We find one primitive root by trying residues $3,5,\ldots$ (2 is out because it is not invertible modulo 22)
  Since $3^5\equiv 1 \pmod{22}$, 3 is not a primitive root modulo 22.
  Since $5^5\equiv 1 \pmod{22}$, 5 is not a primitive root modulo 22.
  Since $7^2\not\equiv 1$, $7^5\equiv -1\not\equiv 1 \pmod{22}$, 7 is a primitive root modulo 22.
- $7^x \pmod{22}$ has order $\frac{10}{\gcd(10,x)}$. We have $\gcd(10,x)=1$ for $x=1,3,7,9$.
- Hence, the primitive roots modulo 22 are $7^1=7, 7^3\equiv 13, 7^7\equiv 17, 7^9\equiv 19$.

Proceeding as in the previous example, we obtain the following result.

Theorem 154. (number of primitive roots) Suppose there is a primitive root modulo n. Then there are $\phi(\phi(n))$ primitive roots modulo n.

Proof. Let x be a primitive root. It has order $\phi(n)$. All other invertible residues are of the form x^a.
Recall that x^a has order $\frac{\phi(n)}{\gcd(\phi(n),a)}$. This is $\phi(n)$ if and only if $\gcd(\phi(n),a)=1$. There are $\phi(\phi(n))$ values a among $1,2,\ldots,\phi(n)$, which are coprime to $\phi(n)$.
In conclusion, there are $\phi(\phi(n))$ primitive roots modulo n. □

Comment. Recall that, for instance, there is no primitive root modulo 15. That's why we needed the assumption that there should be a primitive root modulo n (which is the case if and only if n is of the form $1,2,4,p^k,2p^k$ for some odd prime p).

In particular, since there are always primitive roots modulo primes, we have the following important case:

There are $\phi(\phi(p))=\phi(p-1)$ primitive roots modulo a prime p.

Armin Straub
straub@southalabama.edu

Back to RSA

(RSA encryption)

- Bob chooses large random primes p, q.
- Bob chooses e, and then computes d such that $de \equiv 1 \pmod{(p-1)(q-1)}$.
- Bob makes $N = pq$ and e public. His (secret) private key is d.
- Alice encrypts $c = m^e \pmod{N}$.
- Bob decrypts $m = c^d \pmod{N}$.

Does decryption always work? What Bob computes is $c^d \equiv (m^e)^d = m^{de} \pmod{N}$. It follows from Euler's theorem and $de \equiv 1 \pmod{\phi(N)}$ that $m^{de} \equiv m \pmod{\phi(N)}$ for all invertible residues m. That this actually works for all residues can be seen from the Chinese Remainder Theorem (see Theorem 155 below).

Is that really secure? Well, if implemented correctly (we will discuss potential issues), RSA has a good track record of being secure. Next class, we will actually prove that finding the secret key d is as difficult as factoring N (which is believed, but has not been proven, to be hard). On the other hand, it remains an important open problem whether knowing d is actually necessary to decrypt a given message.

Comment. The $(p-1)(q-1)$ in the generation of d can be replaced with $\operatorname{lcm}(p-1, q-1)$. This will be illustrated in Example 159.

Theorem 155. Let $N = pq$ and d, e be as in RSA. Then, for any m, $m \equiv m^{de} \pmod{N}$.

Comment. Using Euler's theorem, this follows immediately for residues m which are invertible modulo N. However, it then becomes tricky to argue what happens if m is a multiple of p or q.

Proof. By the CRT, we have $m \equiv m^{de} \pmod{N}$ if and only if $m \equiv m^{de} \pmod{p}$ and $m \equiv m^{de} \pmod{q}$.

Since $de \equiv 1 \pmod{(p-1)(q-1)}$, we also have $de \equiv 1 \pmod{p-1}$. By little Fermat, it follows that $m^{de} \equiv m \pmod{p}$ for all $m \not\equiv 0 \pmod{p}$. On the other hand, if $m \equiv 0 \pmod{p}$, then this is obviously true. Thus, $m \equiv m^{de} \pmod{p}$ for all m. Likewise, modulo q. □

Example 156. Bob's public RSA key is $N = 33$, $e = 3$.

(a) Encrypt the message $m = 4$ and send it to Bob.

(b) Determine Bob's secret private key d.

(c) You intercept the message $c = 31$ from Alice to Bob. Decrypt it using the secret key.

Solution.

(a) The ciphertext is $c = m^e \pmod{N}$. Here, $c \equiv 4^3 = 64 \equiv 31 \pmod{33}$. Hence, $c = 31$.

(b) $N = 3 \cdot 11$, so that $\phi(N) = 2 \cdot 10 = 20$.
To find d, we need to compute $e^{-1} \pmod{20}$. Since the numbers are so simple we see $3^{-1} \equiv 7 \pmod{20}$. Hence, $d = 7$.

(c) We need to compute $m = c^d \pmod{N}$, that is, $m = 31^7 \equiv (-2)^7 \equiv 4 \pmod{33}$.
That is, $m = 4$ (as we already knew from the first part).

Example 157. For his public RSA key, Bob needs to select p, q and e. Which of these must be chosen randomly?

Solution. The primes p and q must be chosen randomly. Anything that makes these primes more predictable, makes it easier for an attacker to get her hands on them [in which case, the secret key d is trivial to compute].

On the other hand, e does not need to be chosen at random. In fact, knowing any pair e, d such that $ed \equiv 1 \pmod{(p-1)(q-1)}$ would allow us to factor $N = pq$ (and thus break RSA). We'll prove that later.

Review. RSA

Example 158. If $N=77$, what is the smallest (positive) choice for e?

Solution. Technically, $e=1$ works but then we wouldn't be encrypting at all.
Note that e must be invertible modulo $\phi(N)=6\cdot 10=60$. Hence, $e=2,3,4,5,6$ are not allowed.
The smallest possible choice for e therefore is $e=7$.

Example 159. Bob's public RSA key is $N=33$, $e=13$. His private key is $d=17$.

(a) Explain how the decryption of, say, $c=26$ can be sped up using the CRT.

(b) Encrypt the message $m=4$ and send it to Bob. Compare with the example from last class where $N=33$, $e=3$.

(c) Bob's choice of $e=13$ is actually functionally equivalent to $e=3$ and, similarly, d can be obtained as $e^{-1} \pmod{10}$, resulting in $d=7$. Explain and generalize these claims!

(d) An RSA user is shocked by the previous part and exclaims "RSA is only half as secure as I thought...!" How shocked should we be?

Solution. Note that the private key is $d\equiv 13^{-1} \pmod{20}\equiv 17$.

(a) To decrypt, Bob needs to compute $m=c^d \pmod N$. Knowing that $N=pq=3\cdot 11$, we instead compute $c^d \pmod p$ and $c^d \pmod q$ [which is less work] and then use the CRT to recover $m \pmod N$.
Here, $26^{17}\equiv(-1)^{17}\equiv 2 \pmod 3$ and $26^{17}\equiv 4^{17}\equiv 4^7\equiv 4\cdot 4^2\cdot 4^4\equiv 4\cdot 5\cdot 3\equiv 5 \pmod{11}$.
Hence, $m=26^{17} \pmod{33}\equiv 2\cdot 11\cdot (11)^{-1}_{\bmod 3}+5\cdot 3\cdot (3)^{-1}_{\bmod 11}\equiv 22\cdot(-1)+15\cdot 4\equiv 5 \pmod{33}$.
Comment. Note that $(11)^{-1}_{\bmod 3}$ and $(3)^{-1}_{\bmod 11}$ can be precomputed and reused. In practice, using the CRT leads to about a 4-fold speed up.

(b) The ciphertext is $c=m^e \pmod N$. Here, $c\equiv 4^{13}\equiv\ldots\equiv 31 \pmod{33}$.
If $e=3$ instead, then $c\equiv 4^3=64\equiv 31 \pmod{33}$ so that we get the same ciphertext. See next item!

(c) If you look back at our proof of Theorem 155, you'll see that (again using the CRT) we only need $de\equiv 1 \pmod{(p-1)}$ and $de\equiv 1 \pmod{(q-1)}$ in order that $m^{de}\equiv m \pmod{pq}$.
So, instead of $d\equiv e^{-1} \pmod{(p-1)(q-1)}$, it is enough that $d\equiv e^{-1} \pmod{\operatorname{lcm}(p-1,q-1)}$.
Here, $\operatorname{lcm}(2,10)=10$, so that we only need $d=e^{-1} \pmod{10}$.

(d) It is definitely misleading that RSA is "half" as secure. It is indeed the case though that the key space for the secret key d is only half (or even less) as big as that RSA user initially thought.
However, that means that, for instance, if N is 2048 bit, then the secret key is one bit (possibly more) less than what the shocked RSA user expected. That hardly qualifies as "half as secure".
Comment. However, if $\operatorname{lcm}(p-1,q-1)$ is "too small", that is, $\gcd(p-1,q-1)$ is "too big" (so that we are loosing considerably more than 1 bit for the key size), then p, q should be discarded. If $\gcd(p-1,q-1)\approx 2^e$, then we are loosing about e bits for the key size.

Example 160. RSA is so cool! Why do we even care about, say, AES anymore?

Solution. RSA is certainly cool, but it is very slow (comparatively). As such, RSA is not practical for encrypting larger amounts of data. RSA is, however, perfect for sharing secret keys, which can then be used for encrypting data using, say, AES.

Armin Straub
straub@southalabama.edu

Example 161. Is it a problem that $m=1$ is always encrypted to $c=1$? (Likewise for $m=0$.)

Solution. Well, it would be a problem if we reply to questions using YES (say, 1) and NO (say, 0) and encrypt our reply. However, this would always be a terrible idea in any deterministic public key cryptosystem (that is, a system, in which a message gets encrypted in a single way)!

Why? That's because Eve can just encrypt both YES and NO (or any collection of expected messages) and see which matches the ciphertext she intercepted.

Important conclusion. We must not send messages taken from a small predictable set and encrypt them using a deterministic public key cryptosystem like RSA.

Once realized, this is easy to fix: for instance, Alice can just augment the plaintext with some random garbage in such a way that Bob can discard that garbage after decryption. This is done when RSA is used in practice.

Comment. This applies to any public key cryptosystem, in which a message gets encrypted in a single way. To avoid this issue, some randomness is typically introduced. For instance, for RSA, when used in practice, the plaintext would be padded with random noise before encryption. On the other hand, the ElGamal encryption we discuss next, has such randomness already built into it.

Comment. Note that this is not an issue with symmetric ciphers like DES or AES. In that case, even if the attacker knows that the plaintext must be one of "0" or "1", she still cannot draw any conclusions from intercepting the ciphertext.

Example 162. (extra) Bob's public RSA key is $N=55$, $e=7$.

(a) Encrypt the message $m=8$ and send it to Bob.

(b) Determine Bob's secret private key d.

(c) You intercept the message $c=2$ from Alice to Bob. Decrypt it using the secret key.

Solution.

(a) The ciphertext is $c=m^e \pmod N$. Here, $c\equiv 8^7 \pmod{55}$
$8^2\equiv 9$, $8^4\equiv 9^2\equiv 26$. Hence, $8^7=8^4\cdot 8^2\cdot 8\equiv 26\cdot 9\cdot 8\equiv 2 \pmod{55}$. Hence, $c=2$.

(b) $N=5\cdot 11$, so that $\phi(N)=4\cdot 10=40$.
To find d, we compute $e^{-1} \pmod{40}$ using the extended Euclidean algorithm:

$$\begin{aligned} & \gcd(7,40) && \boxed{40} &= 6\cdot\boxed{7}-2 \\ = \ & \gcd(2,7) && \boxed{7} &= 3\cdot\boxed{2}+1 \\ = \ & 1 \end{aligned}$$

Backtracking through this, we find that Bézout's identity takes the form

$$1=\boxed{7}-3\cdot\boxed{2}=\boxed{7}-3\cdot\left(6\cdot\boxed{7}-\boxed{40}\right)=-17\cdot\boxed{7}+3\cdot\boxed{40}.$$

Hence, $7^{-1}\equiv -17\equiv 23 \pmod{40}$ and, so, $d=23$.

Comment. Actually, as discussed in Example 159, $\phi(N)=(p-1)(q-1)=4\cdot 10$ can be replaced with $\operatorname{lcm}(p-1,q-1)=\operatorname{lcm}(4,10)=20$. It follows that the pair $(e,d)=(7,23)$ is equivalent to the pair $(e,d)=(7,3)$.

(c) We need to compute $m=c^d \pmod N$, that is, $m=2^{23} \pmod{55}$.
$2^2=4$, $2^4=16$, $2^8\equiv 36\equiv -19$, $2^{16}\equiv 19^2\equiv 31 \pmod{55}$. Hence, $2^{23}=2^{16}\cdot 2^4\cdot 2^2\cdot 2\equiv 31\cdot 16\cdot 4\cdot 2\equiv 8 \pmod{55}$.
That is, $m=8$ (as we already knew from the first part).

Comment. As noted above, $d=3$ is equivalent to $d=23$. Indeed, $m=2^3=8 \pmod{55}$.

The ElGamal public key cryptosystem and discrete logarithms

Whereas the security of RSA relies on the difficulty of factoring, the security of ElGamal and Diffie–Hellman relies on the difficulty of computing discrete logarithms.

Discrete logarithms

Suppose $b = a^x \pmod N$. Finding x is called the **discrete logarithm problem** mod N. If N is a large prime p, then this problem is believed to be difficult.

Note. If $b = a^x$, then $x = \log_a(b)$. Here, we are doing the same thing, but modulo N. That's why the problem is called the discrete logarithm problem.

Example 163. Find x such that $4 \equiv 3^x \pmod 7$.

Solution. We have seen in Example 149 that 3 is a primitive root modulo 7. Hence, there must be such an x. Going through the possibilities ($3^2 \equiv 2$, $3^3 \equiv 6$, $3^4 \equiv 4$), we find $x = 4$, because $3^4 \equiv 4 \pmod 7$.

Example 164. Find x such that $3 \equiv 2^x \pmod{101}$.

Solution. Let us check that the solution is $x = 69$. Indeed, a quick binary exponentiation confirms that $2^{69} \equiv 3 \pmod{101}$. (Do it!)

The point is that it is actually (believed to be) very difficult to compute these **discrete logarithms**. On the other hand, just like with factorization, it is super easy to verify the answer if somebody tells us the answer.

Comment. We can check that 2 is a primitive root modulo 101. That is, $2 \pmod{101}$ has (multiplicative) order 100. That means every equation $2^x \equiv a \pmod{101}$, where $a \not\equiv 0$, has a solution.

Diffie–Hellman key exchange

(Diffie–Hellman key exchange)

- Alice and Bob select a large prime p and a primitive root $g \pmod p$.
- Bob randomly selects a secret integer x and reveals $g^x \pmod p$ to everyone.
  Alice randomly selects a secret integer y and reveals $g^y \pmod p$ to everyone.
- Alice and Bob now share the secret $g^{xy} \pmod p$.
  Indeed, Alice can compute $g^{xy} = (g^x)^y$ using the public g^x and her secret y.
  Likewise, Bob can compute $g^{xy} = (g^y)^x$ using the public g^y and his secret x.

Why is this secure? We need to see why eavesdropping Eve cannot (simply) obtain the secret $g^{xy} \pmod p$. She knows g, g^x, $g^y \pmod p$ and needs to find $g^{xy} \pmod p$. This is the **computational Diffie–Hellman problem** (CDH), which is believed to be hard (it would be easy if we could compute discrete logarithms).

Example 165. You are Eve. Alice and Bob select $p = 53$ and $g = 5$ for a Diffie–Hellman key exchange. Alice sends 43 to Bob, and Bob sends 20 to Alice. What is their shared secret?

Solution. If Alice's secret is y and Bob's secret is x, then $5^y \equiv 43$ and $5^x \equiv 20 \pmod{53}$.
Since we haven't learned a better method, we just compute $5^2, 5^3, \ldots$ until we find 43 or 20:
$5^2 = 25$, $5^3 \equiv 19$, $5^4 \equiv 19 \cdot 5 \equiv -11$, $5^5 \equiv -11 \cdot 5 \equiv -2$, $5^6 \equiv -2 \cdot 5 \equiv -10 \equiv 43 \pmod{53}$.
Hence, Alice's secret is $y = 6$. The shared secret is $20^6 \equiv 9 \pmod{53}$.

Note. We don't need to find Bob's secret. [It is $x = 11$.]

Armin Straub
straub@southalabama.edu

ElGamal encryption

Proposed by Taher ElGamal in 1985

The original paper is actually very readable: https://dx.doi.org/10.1109/TIT.1985.1057074

(ElGamal encryption)

- Bob chooses a prime p and a primitive root $g \pmod p$.
  Bob also randomly selects a secret integer x and computes $h = g^x \pmod p$.
- Bob makes (p, g, h) public. His (secret) private key is x.
- To encrypt, Alice first randomly selects an integer y.
  Then, $c = (c_1, c_2)$ with $c_1 = g^y \pmod p$ and $c_2 = h^y m \pmod p$.
- Bob decrypts $m = c_2 c_1^{-x} \pmod p$.

Why does decryption work? $c_2 c_1^{-x} = (h^y m)(g^y)^{-x} = ((g^x)^y m)(g^y)^{-x} = m \pmod p$

More conceptually, the key idea (featured in Diffie–Hellman) that makes ElGamal encryption work is that Alice (her private secret is y) and Bob (his private secret is x) actually share a secret: g^{xy}

Note that encryption is just multiplying m with the shared secret $h^y = g^{xy}$. Likewise, decryption is division by the shared secret $c_1^x = g^{xy}$.

Comment. For ElGamal, the message space actually is $\{1, 2, ..., p-1\}$. $m = 0$ is not permitted.

That's, of course, no practical issue. For instance, we could simply identify $\{1, 2, ..., p-1\}$ with $\{0, 1, ..., p-2\}$ by adding/subtracting 1.

Comment. p and g don't have to be chosen randomly. They can be reused. In fact, it is common to choose p to be a "safe prime" (see next comment), with specific pre-selected choices listed, for instance, in RFC 3526.

Advanced comment. Note that in order to check whether g is a primitive root modulo p, we need to be able to factor $p-1$, which in general is hard (2 is an obvious factor, but other factors are typically large and, in fact, we need them to be large in order for the discrete logarithm problem to be difficult). It is therefore common to start with a prime n and then see if $2n+1$ is prime as well, in which case we select $p = 2n+1$. Such primes p [primes such that $(p-1)/2$ is prime, too] are called **safe primes** (more later).

On the other hand, g doesn't necessarily have to be a primitive root. However, we need the group generated by g (the elements $1, g, g^2, g^3, ...$) to be large. For more fancy cryptosystems, we can even replace these groups with other groups such as those generated by elliptic curves.

Example 166. Bob chooses the prime $p = 31$, $g = 11$, and $x = 5$. What is his public key?

Solution. Since $h = g^x \pmod p$ is $h \equiv 11^5 \equiv 6 \pmod{31}$, the public key is $(p, g, h) = (31, 11, 6)$.

Comment. Bob's secret key is $x = 5$. In principle, an attacker can compute x from $11^x \equiv 6 \pmod{31}$. However, this requires computing a discrete logarithm, which is believed to be difficult if p is large.

Example 167. Bob's public ElGamal key is $(p, g, h) = (31, 11, 6)$.

(a) Encrypt the message $m = 3$ ("randomly" choose $y = 4$) and send it to Bob.

(b) Determine Bob's private key from his public key.

(c) Using Bob's private key, decrypt $c = (9, 13)$.

Solution.

(a) The ciphertext is $c=(c_1,c_2)$ with $c_1=g^y \pmod p$ and $c_2=h^y m \pmod p$.
Here, $c_1=11^4\equiv 9 \pmod{31}$ and $c_2=6^4\cdot 3\equiv 13 \pmod{31}$. Hence, the ciphertext is $c=(9,13)$.

(b) To find Bob's secret key x, we need to solve $11^x\equiv 6 \pmod{31}$. This yields $x=5$.
(Since we haven't learned a better method, we just try $x=1,2,3,\ldots$ until we find the right one.)

Comment. Alternatively, after having done the first part, we know that $m=c_2c_1^{-x} \pmod p$ takes the form $3=13\cdot 9^{-x} \pmod{31}$, which is equivalent to $9^x=13\cdot 3^{-1}\equiv 25 \pmod{31}$. While this also reveals $x=5$, there is an issue with this approach. Can you see it?
[The issue is that 9 (which is c_1 and could be anything) does not have to be a primitive root. In fact, 9 is not a primitive root modulo 31. Accordingly, $9^x\equiv 25 \pmod{31}$ does not have a unique solution: $x=20$ is another one (and does not correspond to Bob's private key).]

(c) We decrypt $m=c_2c_1^{-x} \pmod p$.
Here, $m=13\cdot 9^{-5}\equiv 3 \pmod{31}$.
Comment. One option is to compute $9^{-1}\equiv 7 \pmod{31}$, followed by $9^{-5}\equiv 7^5\equiv 5 \pmod{31}$ and, finally, $13\cdot 9^{-5}\equiv 13\cdot 5\equiv 3 \pmod{31}$. Another option is to begin with $9^{-5}\equiv 9^{25} \pmod{31}$ (by Fermat's little theorem).

Example 168. (extra) Bob's public ElGamal key is $(p,g,h)=(23,10,11)$.

(a) Encrypt the message $m=5$ ("randomly" choose $y=2$) and send it to Bob.

(b) Encrypt the message $m=5$ ("randomly" choose $y=4$) and send it to Bob.

(c) Break the cryptosystem and determine Bob's secret key.

(d) Use the secret key to decrypt $c=(8,7)$.

(e) Likewise, decrypt $c=(18,19)$.

Solution.

(a) The ciphertext is $c=(c_1,c_2)$ with $c_1=g^y \pmod p$ and $c_2=h^y m \pmod p$.
Here, $c_1=10^2\equiv 8 \pmod{23}$ and $c_2=11^2\cdot 5\equiv 6\cdot 5\equiv 7 \pmod{23}$. Hence, the ciphertext is $c=(8,7)$.

(b) Now, $c_1=10^4\equiv 18 \pmod{23}$ and $c_2=11^4\cdot 5\equiv 13\cdot 5\equiv 19 \pmod{23}$ so that $c=(18,19)$.

(c) To find Bob's secret key x, we need to solve $10^x\equiv 11 \pmod{23}$. This yields $x=3$.
(Since we haven't learned a better method, we just try $x=1,2,3,\ldots$ until we find the right one.)

(d) We decrypt $m=c_2c_1^{-x} \pmod p$.
Here, $m=7\cdot 8^{-3}\equiv 7\cdot 4\equiv 5 \pmod{23}$, as we knew from the first part.
[$8^{-1}\equiv 3 \pmod{23}$, so that $8^{-3}\equiv 3^3\equiv 4 \pmod{23}$. Or, use Fermat: $8^{-3}\equiv 8^{19}\equiv 4 \pmod{23}$.]

(e) In this case, $m=19\cdot 18^{-3}\equiv 19\cdot 16\equiv 5 \pmod{23}$, as we knew from the second part.

Review. ElGamal encryption

- Like RSA, ElGamal is terribly slow compared with symmetric ciphers like AES.

  Encryption under ElGamal requires two exponentiations (slower than RSA); however, these exponentiations are independent of the message and can be computed ahead of time if need be (in that case, encryption is just a multiplication, which is much faster than RSA). Decryption only requires one exponentiation (like RSA).

- In contrast to RSA, ElGamal is randomized. That is, a single plaintext m can be encrypted to many different ciphertexts.

  A drawback is that the ciphertext is twice as large as the plaintext.

  On the positive side, an attacker who might be able to guess potential plaintexts cannot (as in the case of vanilla RSA) encrypt these herself and compare with the intercepted ciphertext.

Example 169. If Bob selects $p = 23$ for ElGamal, how many possible choices does he have for g? Which are these?

Solution. g needs to be a primitive root modulo 23. Recall that, modulo a prime p, there are $\phi(\phi(p)) = \phi(p-1)$ many primitive roots. Hence, Bob has $\phi(p-1) = \phi(22) = 10$ choices for g.

Example 170. Does Alice have to choose a new y if she sends several messages to Bob using ElGamal encryption?

Solution. Yes, she absolutely has to randomly choose a new y every time! Here's why:

If she was using the same y to encrypt messages $m^{(1)}$ and $m^{(2)}$, Alice would be sending the ciphertexts $(c_1^{(1)}, c_2^{(1)}) = (g^y, g^{xy} m^{(1)})$ and $(c_1^{(2)}, c_2^{(2)}) = (g^y, g^{xy} m^{(2)})$.

That means, Eve can immediately figure out $c_2^{(1)} \,/\, c_2^{(2)} = m^{(1)} \,/\, m^{(2)}$ (the divison is a modular inverse and everything is modulo p). That's a combination of the plaintexts, and Eve should never be able to get her hands on such a thing.

(Note that Eve would know right away if Alice is doing the mistake of reusing y because $c_1^{(1)} = c_1^{(2)}$.)

Comment. The situation is just like for the one-time pad (in that case, reusing the key reveals $m^{(1)} \oplus m^{(2)}$).

The computational and decisional Diffie–Hellman problem

We indicated that the security of ElGamal depends on the difficulty of computing discrete logarithms. Here is a more precise statement.

Theorem 171. Obtaining m from c (without the private key) in ElGamal is exactly as difficult as the **computational Diffie–Hellman problem** (CDH).

The CDH problem is the following: given $g, g^x, g^y \pmod p$, find $g^{xy} \pmod p$. It is believed to be hard.

Proof. Recall that the public key is $(p, g, h) = (p, g, g^x)$. The ciphertext is $c = (g^y, h^y m) = (g^y, g^{xy} m)$.

Hence, determining m is equivalent to finding g^{xy}.

Since $g, g^x, g^y \pmod p$ are known, this is precisely the CDH problem. □

Armin Straub
straub@southalabama.edu

Example 172. In fact, even the **decisional Diffie–Hellman problem** (DDH) is believed to be difficult.

The DDH problem is the following: given g, g^x, g^y, $r \pmod p$, decide whether $r \equiv g^{xy} \pmod p$. Obviously, this is simpler than the CDH problem, where g^{xy} needs to be computed. Yet, it, too, is believed to be hard.

Comment. Well, at least it is hard (modulo p) if we always want to do better than guessing.

Here's how we can sometimes do better than guessing: if g^x or g^y are quadratic residues (this is actually easy to check modulo primes p using quadratic reciprocity and the Legendre symbol), then g^{xy} is a quadratic residue (why?!). Hence, if r is not a quadratic residue, we can conclude that $r \not\equiv g^{xy}$.

More on safe primes

Recall that p is a **safe prime** if both p and $(p-1)/2$ are prime. The next example illustrates why it is common to use safe primes for ElGamal.

In general, it is difficult to ensure that g is a primitive root, or almost a primitive root, modulo p.

Example 173. Suppose that p is a safe prime. Show that all residues $g \not\equiv 0, \pm 1 \pmod p$ have order $(p-1)/2$ or $p-1$.

In the latter case, g is a primitive root. In fact, if $p > 5$, then half of the residues $g \not\equiv 0, \pm 1$ are primitive roots.

Solution. Suppose $g \not\equiv 0, \pm 1 \pmod p$. Because p is a prime and $g \not\equiv 0$, g is invertible. Its multiplicative order N divides $\phi(p) = p-1$. But the prime factorization of $p-1$ is 2 times $(p-1)/2$. Hence, the only possible orders are $1, 2, (p-1)/2$ and $p-1$. The residues ± 1 are the only with order 1 and 2 (why?!). Thus, g must have order $(p-1)/2$ or $p-1$.

Finally, if $p > 5$ (so that $(p-1)/2$ is odd), note that the number of primitive roots is $\phi(p-1) = \phi(2)\phi((p-1)/2) = (p-3)/2$, which is exactly half of the residues g.

Advanced comment. Actually, it is easy to distinguish between the residues that have order $(p-1)/2$ and those that have order $p-1$. Recall that, if x has order $p-1$, then x^2 has order $\frac{p-1}{\gcd(p-1,2)} = \frac{p-1}{2}$. It follows that (among the $x \not\equiv 0, \pm 1$) quadratic residues have order $(p-1)/2$. (And, using quadratic reciprocity, it is computationally easy to determine whether a residue modulo p is a quadratic residue or not.)

Example 174. Is there any advantage for RSA if p is a safe prime? Potential issues?

Solution. If p is a safe prime, then $\gcd(p-1, q-1) = 2$. Why?!

Hence, the key space is as large as possible.

On the other hand, we need to think about whether we are weakening the security in case we might severely limit the number of possible p's to choose from.

Another issue is that generating random safe primes is considerably more work. On the other hand, Bob usually does not generate a public key frequently, so that this might not be much of an issue.

Further comments on RSA and ElGamal

Theorem 175. Determining the secret private key d in RSA is as difficult as factoring N.

Proof. Let us show how to factor $N = pq$ if we know e and d.

- Write $ed - 1 = 2^t m$, where t is chosen as large as possible such that 2^t divides $ed - 1$.
  Since $ed - 1 \equiv 0 \pmod{(p-1)(q-1)}$ and 2^2 divides $(p-1)(q-1)$, we have $t \geqslant 2$.
- Pick a random invertible residue x. Observe that $x^{ed-1} \equiv 1 \pmod{N}$. In other words, $(x^m)^{2^t} \equiv 1$.
  Hence, the multiplicative order of x^m must divide 2^t.
- Suppose that x^m has different order modulo p than modulo q.

  Note. This works for at least half of the (invertible) residues x. If we are unlucky, we just select another x.

  Since both orders must divide 2^t, we may suppose x^m has order 2^s modulo p, and larger order modulo q.
  Then, $x^{2^s m} \equiv 1 \pmod{p}$ but $x^{2^s m} \not\equiv 1 \pmod{q}$.
  Consequently, $\gcd(x^{2^s m} - 1, N) = p$ so that we have found the factor p of N.

  Note. Of course, we don't know s (because we don't know p and q), but we can just go through all $s = 1, 2, \ldots, t-1$. One of these has to reveal the factor p. □

However. It is not known whether knowing d is actually necessary for Eve to decrypt a given ciphertext c. This remains an important open problem.

Example 176. (homework) Bob's public RSA key is $N = 323$, $e = 101$. Knowing $d = 77$, factor N using the approach of the previous theorem.

Solution. Here, $de - 1 = 7776 = 2^5 \cdot 243$ so that $t = 5$ and $m = 243$.

- Let's pick $a = 2$. $a^m = 2^{243} \equiv 246 \pmod{323}$ must have order dividing 2^5.
  $\gcd(246^2 - 1, 323) = 19$ (so we don't even need to check $\gcd(246^{2^s} - 1, 323)$ for $s = 2, 3, 4$)
  Hence, we have factored $N = 17 \cdot 19$.

Comment. Among the $\phi(323) = 16 \cdot 18 = 288$ invertible residues a, only 36 would not lead to a factorization. The remaining 252 residues all reveal the factor 19.

Another project idea. Run some numerical experiments to get a feeling for the number of residues that result in a factorization.

Definition 177. Bob's public key cryptosystem is **semantically secure** if Eve cannot do better than guessing in the following challenge:

- Bob determines a random public and private key. The public key is given to Eve.
- Eve selects two plaintexts m_1 and m_2.
- Alice flips a fair coin and, accordingly, using the public key encrypts m_1 or m_2 as c.
- Eve now needs to decide whether c is the encryption of m_1 or m_2.

For this definition to make precise mathematical sense, we need to assume that Eve's computing power is somehow limited (typically, she is limited to polynomial-time algorithms).

Comment. Also, many variations exist of what semantic security exactly is. All of these try to capture the idea that an attacker does not learn anything about m from knowing c. The one above is often referred to as IND-CPA (Indistinguishability under Chosen Plaintext Attack).

Important comment. Realize that semantic security is a very strong property to ask for! In particular, this is much stronger than what we usually think about in terms of security: you might call a cipher secure if it is "impossible" for an attacker to get m from c. Semantic security is requiring that an attacker gets so little information from c that she cannot even tell whether it came from (her own choices) m_1 or m_2.

Example 178. Is vanilla RSA semantically secure?

Solution. No. Eve can just encrypt both m_1 and m_2 herself, and compare with c. She then knows for sure which of the two was encrypted.

Comment. As mentioned before, in practice, RSA is never used in its vanilla (or "textbook") version (unless random plaintexts are encrypted). Instead, it is randomized (like ElGamal is by design) by padding the plaintext with random stuff.

Check out OAEP: https://en.wikipedia.org/wiki/Optimal_asymmetric_encryption_padding

The resulting RSA-OAEP has been proven semantically secure (under the "RSA assumption" that finding m from c is hard).

Example 179. Is ElGamal semantically secure?

Solution. Essentially, yes.

Recall that the public key is $(p, g, h) = (p, g, g^x)$.

The ciphertext is $(c_1, c_2) = (g^y, h^y m) = (g^y, g^{xy} m)$. Eve needs to decide whether the m in there is m_1 or m_2.

Equivalently, she needs to decide whether $r = c_2/m_1$ (or $r = c_2/m_2$) equals g^{xy} or not.

This is essentially the DDH problem.

Strictly speaking. Because of the issue with quadratic residues mentioned when we introduced the DDH problem, ElGamal is not semantically secure in the sense we defined things. However, if we wanted (this is more of a theoretical point), this issue could be fixed by not computing with all invertible residues modulo p, but only with quadratic residues. We could further select p to be a **safe prime**, meaning that $(p-1)/2$ is prime again, in which case all quadratic residues (except 1) have order $(p-1)/2$ (so that no similar games can be played using orders of elements).

Practical implications. Indeed, Diffie–Hellman and ElGamal in practice often use safe primes p. In that case, as we observed in Example 173, there are no elements of small order (besides 1 and -1). Since generating such primes can be a bit expensive, it is common to use preselected ones. For instance, RFC 3526 lists six such primes (together with a generator g) with $1536, 2048, \ldots, 8192$ bits.

https://www.ietf.org/rfc/rfc3526.txt

Important. It is perfectly fine that p and g are not random in Diffie–Hellman or ElGamal. However, it is absolutely crucial that x (and y) are random (generated using a cryptographically secure PRG).

Example 180. What is your feeling? Can we make RSA even more secure by allowing N to factor into more than 2, say, 3 primes?

Solution. That doesn't seem like a good idea. Namely, observe that the security of RSA relies on adversaries being unable to factor N. Allowing more factors of N (while keeping the size of N fixed) makes that task easier, because more factors means that the factors are necessarily smaller.

Example 181. RSA has proven to be secure so far. However, it is easy to implement RSA in such a way that it is insecure. One important but occasionally messed up part of RSA is that **p and q must be unpredictable**, and the only way to achieve that is to choose p, q completely randomly in some huge interval $[M_1, M_2]$.

- For instance, if $N = pq$ has m digits and we know the first (or last) $m/4$ digits of p, then we can efficiently factor N.

  An adversary might know many digits of p if, for instance, we make the mistake of generating the random prime p by considering candidates of the form $10^{100} + k$ for small (random) values of k (10^{100} has no special significance; it can be replaced with any large number).

- Also, we must use a cryptographically secure PRG to generate p and q.

  If using a "bad" PRG or choosing seeds with too little entropy, then (especially among a large number of public keys generated this way) it becomes likely that (different) public keys N and N' share a prime factor p. In that case, everybody can determine $p = \gcd(N, N')$ and break both public keys.

  Indeed. For instance, in a study of Lenstra et. al., millions of public keys were collected and compared. Among the RSA moduli, about 0.2% shared a common prime factor with another one. That's terrible: if (different) public keys N and N' share a prime factor p, then everybody can determine $p = \gcd(N, N')$ and break both public keys.

  http://eprint.iacr.org/2012/064.pdf

- In that direction, is the security of public key cryptosystems like RSA in any way compromised when used by tens of millions of users?

  As noted above, millions of people using "bad" PRGs for generating RSA public keys make it likely that this weakness can be practically exploited.

  Similarly, for Diffie–Hellman and ElGamal, it is common to use fixed primes p. While fine in principle, this may be an issue if used by millions of users faced against an adversary Eve with vast resources. See, for instance: https://threatpost.com/prime-diffie-hellman-weakness-may-be-key-to-breaking-crypto/

Example 182. (side-channel attacks) For instance, by measuring the time it takes to decrypt messages as $m = c^d \pmod N$ in RSA, Eve might be able to reconstruct the secret key d.

This **timing attack**, first developed by Paul Kocher (1997), is particularly unsettling because it illustrates that the security of a system can be compromised even if mathematically everything is sound. This sort of attack is called a **side-channel attack**. It attacks the implementation (software and/or hardware) rather than the cryptographic algorithm.

See Section 6.2.3 in our book for more details on how d can be obtained in this attack.

In a similar spririt, there exist power attacks (measuring power instead of time during decryption) or fault attacks (for instance, injecting errors during computations):

https://en.wikipedia.org/wiki/Side-channel_attack

How to prevent? Implement RSA in such a way that no inferences can be drawn from the time and power consumption.

Lesson. Do not implement crypto algorithms yourself!! Instead, use one of the well-tested open implementations.

It's kind of sad, isn't it? Don't come up with your own ciphers. Don't implement ciphers yourself...

But it is important to realize just how easy it is to implement these algorithms in such a way that security is compromised (even if the idea, intentions and algorithms are all sound and secure).

After advertising open implementations, let us end this discussion with a cautionary example in that regard.

Example 183. The following story made lots of headlines in 2016:

https://threatpost.com/socat-warns-weak-prime-number-could-mean-its-backdoored/116104/

After a year, it was noticed that, in the open-source tool Socat ("Netcat++"), the Diffie-Hellman key exchange was implemented using a hard-coded 1024 bit prime p (nothing wrong with that), which wasn't prime! Explain how this could be used as a backdoor.

Solution. The security of the Diffie-Hellman key exchange relies on the difficulty of taking discrete logarithms modulo p. If we can compute x in $h = g^x \pmod p$, then we can break the key exchange.

Now, if $p = p_1 p_2$, then we can use the CRT to find x by solving the two (much easier!) discrete logarithm problems

$$h = g^x \pmod{p_1}, \quad h = g^x \pmod{p_2}.$$

This is an example of a **NOBUS backdoor** ("nobody but us"), because the backdoor can only be used by the person who knows the (secret) factorization of p.

Comment. In the present case, the Socat "prime" p actually has the two small factors 271 and 13597, and $p/(271 \cdot 13597)$ is still not a prime (but nobody has been able to factor it). This might hint more at a foolish accident than a malicious act.

Important follow-up question. Of course, the issue has been fixed and the composite number has been replaced by the developers with a large prime. However, should we trust that it really is a prime?

We don't need to trust anyone because primality checking is simple! We can just run the Miller–Rabin test N times. If the number was composite, there is only a 4^{-N} chance of us not detecting it. (In OpenSSL, for instance, $N = 40$ and the chance for an error, 2^{-80}, is astronomically low.) Both Fermat and Miller–Rabin instantly detect the number here to be composite (for certain).

Comment. This illustrates both what's good and what's potentially problematic about open source projects. The potentially problematic part for crypto is that Eve might be among the people working on the project. The good part is that (hopefully!*) many experts are working on or looking into the code. Thus, hopefully, any malicious acts on Eve's part should be spotted soon (in fact, with proper code review, should never make it into any production version). Of course, this "hope" requires ongoing effort on the parts of everyone involved, and the willingness to fund such projects.

*However, sometimes very few people are involved in a project, despite it being used by millions of users. For instance, see: https://en.wikipedia.org/wiki/Heartbleed

Example 184. (short plaintext attack on RSA) Suppose a 56bit DES key (or any other short plaintext) is written as a number $m \approx 2^{56} \approx 10^{16.9}$ and encrypted as $c = m^e \pmod N$.

Eve makes two lists:

- $cx^{-e} \pmod N$ for $x = 1, 2, ..., 10^9$
- $y^e \pmod N$ for $x = 1, 2, ..., 10^9$

If there is a match between the lists, that is $cx^{-e} = y^e \pmod N$, then $c = (xy)^e \pmod N$ and Eve has learned that the plaintext is $m = xy$.

This attack will succeed if m is the product of two integers x, y (up to 10^9). This is the case for many integers m.

Another project idea. Quantify how many integers factor into two small factors.

How to prevent? To prevent this attack, the plaintext can be padded with random bits before being encrypted. Recall that we should actually never use vanilla RSA (unless with random plaintexts) and always use a securely padded version instead!

Example 185. For RSA, does double (or triple) encryption improve security?

(a) Say, if Bob asks people to send him messages first encrypted with a first public key (N, e_1) and then encrypted with a second public key (N, e_2).

(b) Or, what if Bob asks people to send him messages first encrypted with a first public key (N_1, e_1) and then encrypted with a second public key (N_2, e_2).

Solution.

(a) No, this does not result in any additional security.
After one encryption, $c_1 = m^{e_1} \pmod N$ and the final ciphertext is $c_2 = c_1^{e_2} \pmod N$. However, note that $c_2 = m^{e_1 e_2} \pmod N$, which is the same as encryption with the single public key $(N, e_1 e_2)$.

(b) This adds only a negligible bit of security and hence is a bad idea as well. The reason is that an attacker able to determine the secret key for (N_1, e_1) is likely just as able to determine the secret key for (N_2, e_2), meaning that the attack would only take twice as long (or two computers). That's only a tiny bit of security gained, somewhat comparable to increasing N from 1024 to 1025 bits. If heightened security is wanted, it is better to increase the size of N in the first place.
[Make sure you see how the situation here is different from the situation for 3DES.]

Example 186. (common modulus attack on RSA) Alice encrypts m using each of the RSA public keys $(N,\ e_1)$ and $(N,\ e_2)$ so that the ciphertexts are $c_1 = m^{e_1} \pmod N$ and $c_2 = m^{e_2} \pmod N$. Eve might be able to figure out m from c_1 and c_2!! How and when?

Solution. The crucial observation is that $c_1^x c_2^y \equiv m^{e_1 x} m^{e_2 y} = m^{e_1 x + e_2 y} \pmod N$. Eve can choose x and y.
She knows m if she can arrange x and y such that $e_1 x + e_2 y = 1$. This is possible if $\gcd(e_1, e_2) = 1$, in which case Eve would use the extended Euclidean algorithm to determine appropriate x and y.

A scenario. Bob's public RSA key is (N, e). However, when Alice requests this public key from Bob, her message gets intercepted by Eve who instead sends (N, e_2) back to Alice, where e_2 differs from e in only one bit. Alice uses (N, e_2) to encrypt her message and sends c_2 to Bob. Of course, Bob fails to decrypt Alice's message and so resends his public key to Alice (this time, Eve doesn't intervene). Alice now uses (N, e) to encrypt her message and send c to Bob.

Since $e - e_2 = \pm 2^r$, we have $\gcd(e, e_2) = 1$ (why?!), so that Eve can determine m as explained above.

Comment on that scenario. From a practical point of view, we can argue that, if Eve can trick Alice into using a modified version of Bob's public key, then she might as well give a completely new public key (that Eve created) to Alice, in which case she can immediately decipher c_2. That's certainly true. However, that way, Eve's malicious intervention would be plainly visible as such.

Example 187. (chosen ciphertext attack on RSA) Show that RSA is not secure under a chosen ciphertext attack.

First of all, let us recall that in a chosen ciphertext attack, Eve has some access to a decryption device. In the present case, we mean the following: Eve is trying to determine m from c. Clearly, we cannot allow her to use the decryption device on c (because then she has m and nothing remains to be said). However, Eve is allowed to decrypt some other ciphertext c' of her choosing (hence, "chosen ciphertext").

You may rightfully say that this is a strange attacker, who can decrypt messages except the one of particular interest. This model is not meant to be realistic; instead, it is important for theoretical security considerations: if our cryptosystem is secure against this (adaptive) version of chosen ciphertext attacks, then it is also secure against any other reasonable chosen ciphertext attacks.

Solution. RSA is not secure under a chosen ciphertext attack:

Suppose $c = m^e \pmod N$ is the ciphertext for m.

Then, Eve can ask for the decryption m' of $c' = 2^e c \pmod N$. Since $c' = (2m)^e \pmod N$, Eve obtains $m' \equiv 2m$, from which she readily determines $m = 2^{-1} m' \pmod N$.

Comment. On the other hand, RSA-OAEP is provably secure against chosen ciphertext attacks. Recall that, in this case, m is padded prior to encryption. As a result, $2m$ or, more generally am, is not going to be a valid plaintext.

Example 188. What we just exploited is that RSA is **multiplicatively homomorphic**.

Multiplicatively homomorphic means the following: suppose m_1 and m_2 are two plaintexts with ciphertexts c_1 and c_2. Then, (the residue) $m_1 m_2$ has ciphertext $c_1 c_2$.

[That is, multiplication of plaintexts translates to multiplication of ciphertexts, and vice versa. Mathematically, this means that the map $m \to c$ is a homomorphism (with respect to multiplication).]

Indeed, for RSA, $c_1 = m_1^e$ and $c_2 = m_2^e$, so that $c_1 c_2 = m_1^e m_2^e = (m_1 m_2)^e \pmod N$ is the ciphertext for $m_1 m_2$.

Why care? In our previous example, being multiplicatively homomorphic was a weakness of RSA (which is "cured" by RSA-OAEP). However, there are situations where homomorphic ciphers are of practical interest. With a homomorphic cipher, we can do calculations using just the ciphertexts without knowing the plaintexts (for instance, the ciphertexts could be encrypted (secret) votes, which could be publicly posted; then anyone could add up (in an additively homomorphic system) these votes into a ciphertext of the final vote count; the advantage being that we don't need to trust an authority for that count). The search for a fully **homomorphic encryption** scheme is a hot topic. For a nice initial read, you can find more at:

https://blog.cryptographyengineering.com/2012/01/02/very-casual-introduction-to-fully/

Example 189. (chosen ciphertext attack on ElGamal) Show that ElGamal is not secure under a chosen ciphertext attack.

Solution. Recall, again, that in a chosen ciphertext attack, Eve is trying to determine m from c and Eve has access to a decryption device, which she can use, except not to the ciphertext c in question.

Suppose $c = (c_1, c_2) = (g^y, g^{xy} m)$ is the ciphertext for m. Then $(c_1, 2c_2) = (g^y, g^{xy} 2m)$ is a ciphertext for $2m$. Hence, Eve can ask for the decryption of $c' = (c_1, 2c_2)$, which gives her $m' = 2m$, from which she determines $m = 2^{-1} m' \pmod p$.

In fact, again, the reason that ElGamal is not secure under a chosen ciphertext attack is that it is multiplicatively homomorphic.

Example 190. Show that ElGamal is multiplicatively homomorphic.

Solution. Let $(g^{y_1}, g^{xy_1} m_1)$ be a ciphertext for m_1, and $(g^{y_2}, g^{xy_2} m_2)$ a ciphertext for m_2.

The product (component-wise) of the ciphertexts is $(g^{y_1+y_2}, g^{x(y_1+y_2)} m_1 m_2)$, which is a ciphertext for $m_1 m_2$. So, again, the product of ciphertexts corresponds to the product of plaintexts.

A quick summary of some aspects of RSA and ElGamal.

- As long as appropriate key sizes are used, both RSA and ElGamal appear secure.

  About the same key size needed for both: at least 1024 bits. By now, better 2048 bits.

- The security of both RSA and ElGamal can be compromised by using a cryptographically insecure PRG to generate the secret pieces p, q (for RSA) or x (for ElGamal).

- It is important to have different ciphers, especially ones that rely on the difficulty of different mathematical problems.

  Comment. Factoring $N = pq$ and computing discrete logarithms modulo p are the two different problems for RSA and ElGamal, respectively. It is not known whether the ability to solve one of them would make it significantly easier to also solve the other one. However, historically, advances in factorization methods (like the number field sieve) have subsequently lead to similar advances in computing discrete logarithms. Both problems seem of comparable difficulty.

- Both are multiplicatively homomorphic, but RSA loses this property when padded.