Extended Introduction to Computer Science
CS1001.py

Chapter F Topics in Number Theory:
Lecture 13 Diffie-Hellman Secret Key Exchange

Michal Kleinbort, Amir Rubinstein

School of Computer Science
Tel-Aviv University
Fall Semester 2023-24
http://tau-cs1001-py.wikidot.com

* Slides based on a course designed by Prof. Benny Chor

Topics in Number Theory: Plan

1. Exponentiation of integers

2. Primality testing (using Fermat's "little theorem")

3. Diffie-Helman secret key exchange

4. Euclid's GCD (greatest common divisor)

Encryption: Basic Model

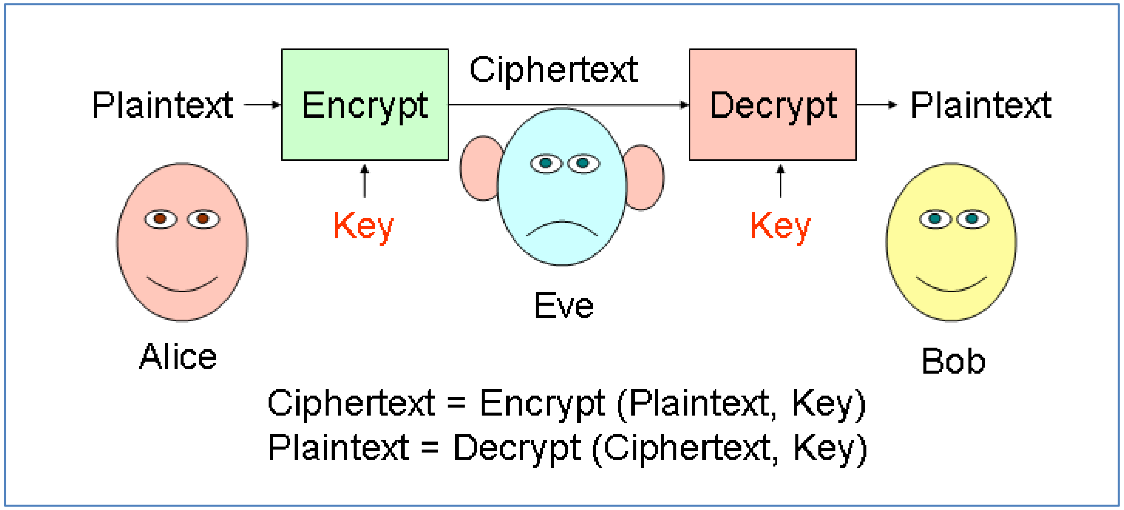
- Let us welcome the three major players in this field, Alice, Bob and Eve:

- Alice and Bob communicate over an insecure channel (anyone, in particular Eve, can eavesdrop).

- Goal: send a message from Alice to Bob confidentially (so Eve will not understand it).

Encryption: Basic Model (2)

- Alice wants to send Bob some message, called plaintext.
- She encrypts the plaintext, using an encryption algorithm, which employs a secret value called key. The encrypted message is called the ciphertext.
- Bob receives the ciphertext, and employs a decryption algorithm with the same key as Alice used, to get the original plaintext.



Credit: https://www.cs.virginia.edu/~evans/dragoncrypto/daylmth.2

- Eve knows the ciphertext, encryption and decryption algs, but not the secret key, without which decryption is computationally hard.

Toy Example: Caesar Encryption

- Named after Julius Caesar, who used it to protect messages of military significance.
- Replace the alphabet letters in a cyclic manner, using some fixed offset.
- What is the secret key here?

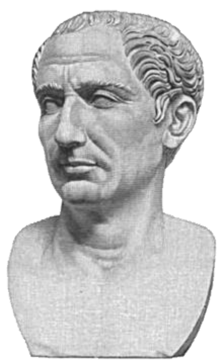


Image from Wikipedia

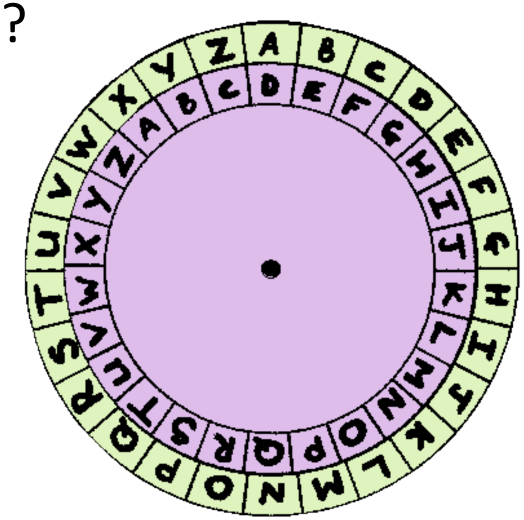


Image from:
http://www.maths-resources.net

Key (offset) = +3

- This is a toy example, since breaking the encryption is easy in this case. Simply check all possible offsets and see which yields a meaningful text.

The Key Exchange Problem

- Additional encryption methods have been used over the years. One famous example is the German Enigma Machine, which utilized a new key each day.

From Wikipedia

- However, this is not the topic of this lecture. We will deal today with the problem of key exchange: Alice and Bob need to share the same secret key, which must be secretly generated and exchanged prior to using the insecure channel for communication.

- A major problem, especially at the internet era: How can Alice and Bob secretly generate and exchange a key, even if they have never physically met, they live on antipodal sides of the globe, and all communication lines are insecure (subject to eavesdropping)?

Diffie Hellman Key Exchange (1976)

- The basic idea: use a one-way function.
- This is a function that is easy (computationally) to compute in one direction, but hard (again, computationally) in the reversed direction.

(figures from Wikipedia)

Color Mixing Analogy

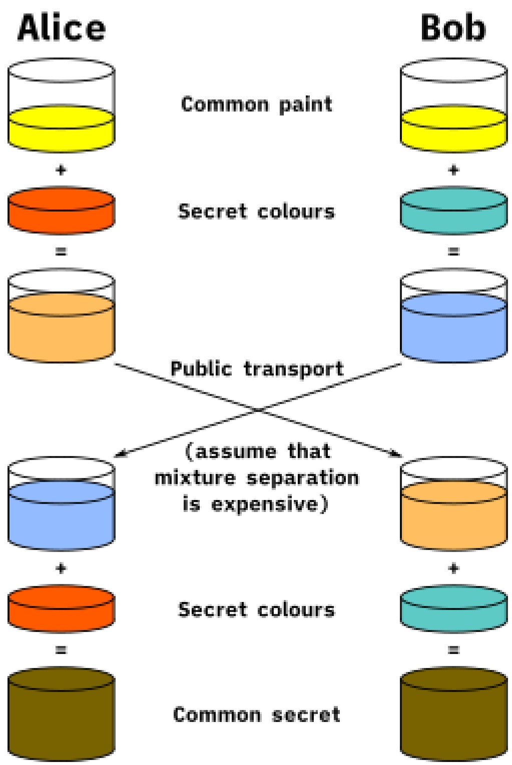


Figure from Wikipedia

See also this video: https://youtu.be/YEBfamv-_do?t=144 (DH starts at 2:25)

Discrete Log: A One-way Function

- Let p be a large prime (say 1024 bits long).
- Let g be a random integer in the range $1 < g < p - 1$.
- Let $x = g^i \bmod p$ for some $1 \leq i < p - 1$.

- The inverse operation,
  $x = g^i \mod p \mapsto i$ (called discrete log) is believed to be computationally hard.
- We say that the mapping $i \longrightarrow g^i \mod p$ is a one way function.
- This is a computational notion. With unbounded (or even just exponential) resources, one can invert this function (compute discrete log).
- Note: computing (non-descrete) log is easy (but we do not show this).

Modular Exponentiation Properties

Questions about the order of exponentiation and mod p operations are often raised.

Well, all the following hold (we are interested in the last one for our purposes):

- $((a \mod p) + (b \mod p)) \mod p = (a + b) \mod p$.
- $((a \mod p) \cdot (b \mod p)) \mod p = (a \cdot b) \mod p$.
- $(g^a \mod p)^b \mod p = (g^a)^b \mod p$.

In fact, all these mod p operations are best viewed in the context of the finite field Z_p^* (learned in algebra courses).

Diffie Hellman Key Exchange (1976)

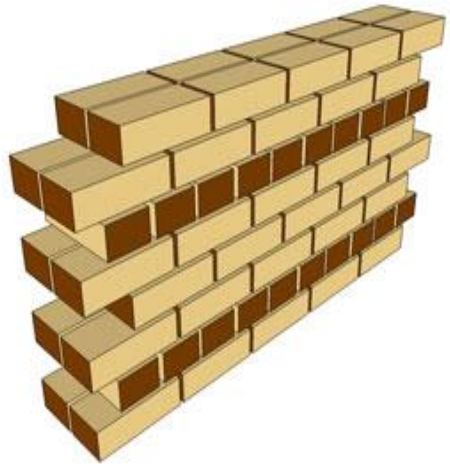
- Public parameters: A large prime p (1024 bit long, say) and a random element g in in the range $1 < g < p-1$.
- Alice chooses at random an integer a from the interval $[2..p-2]$. She sends $x = g^a \pmod{p}$ to Bob (over the insecure channel).
- Bob chooses at random an integer b from the interval $[2..p-2]$. He sends $y = g^b \pmod{p}$ to Alice (over the insecure channel).

- Alice, holding a, computes $y^a = (g^b)^a = g^{ba} \pmod{p}$.
- Bob, holding b, computes $x^b = (g^a)^b = g^{ba} \pmod{p}$.
- Now both have the shared secret, $g^{ba} \pmod{p}$.
- An eavesdropper cannot infer the key, $g^{ba} \pmod{p}$ after seeing "only" $p,\ g,\ x = g^a \pmod{p}$ and $y = g^b \pmod{p}$ (under the assumption that discrete log is intractable).

- We have just witnessed a small miracle !

Diffie Hellman Key Exchange (1976)

Public: Large prime *p*, and some *g* $(1<g<p)$



<u>Alice</u>

Secret: random *a*
$(1< a <p)$

<u>Bob</u>

Secret: random *b*
$(1< b <p)$

$x = g^a \bmod p$

computation (one pow each)

$y = g^b \bmod p$

Communication over insecure channels (one mssg each)

$y^a \bmod p$
$= (g^b \bmod p)^a \bmod p$
$= g^{ab} \bmod p$

computation (one pow each)

$x^b \bmod p$
$= (g^a \bmod p)^b \bmod p$
$= g^{ab} \bmod p$

Diffie Hellman Key Exchange in Python

- We show a centralized simulation of DH:

```
def DH_exchange():
    """ generates a shared DH key """
    n = int(input("How many bits for the prime number? "))
    p = find_prime(n)
    print("p =",p, "a large prime")
    g = random.randint(2,p-1)
    print("g =",g, "random 1<g<p")
    print()
    a = random.randint(2,p-1)# Alice's  secret
    print("a = ? random secret of Alice")
    b = random.randint(2,p-1)# Bob's  secret
    print("b = ? random secret of Bob")
    print()
    x = pow(g,a,p) #Alice's transmission
    print("x =",x, "Alice sends to Bob x = g**a%p")
    y = pow(g,b,p) #Bob's transmission
    print("y =",y, "Bob sends to Alice y = g**b%p")
    print()
    key_A = pow(y,a,p) #shared key on Alice's side
    print("key_A =", key_A, "shared key on Alice's side y**a%p")
    key_B = pow(x,b,p) #shared key on Bob's side
    print("key_B =", key_B, "shared key on Bob's side x**b%p")
    if key_A != key_B:
        print("This can't happen!", key_A, "!=", key_B)
```

Diffie Hellman Key Exchange in Python

```
>>> DH_exchange()
How many bits for the prime number? 3
p = 5 a large prime
g = 3 random 1<g<p

a = ? random secret of Alice
b = ? random secret of Bob

x = 2 Alice sends to Bob x = g**a%p
y = 1 Bob sends to Alice y = g**b%p

key_A = 1 shared key on Alice's side y**a%p
key_B = 1 shared key on Bob's side x**b%p
```

Diffie Hellman Key Exchange in Python

```
>>> DH_exchange()
How many bits for the prime number? 512
P = 7640895672576680265081623351953774950427012566126772551005189 4
    3019177774114418822422098382023396705281980925359026966022353 1
    18651788567116015502596244275 3 a large prime
g = 7583379885198628449195713447518122655417408786461100036574044 4
    2662628948047374071414661893074070548650243173871657298446277 3
    95444079861704166114403644000 8 random 1<g<p

a = ? random secret of Alice
b = ? random secret of Bob

x = 5291546581133049668902094269662962207235754173713553344158039 0
    2784746790996301977648897065802098119152763083158475110648534 1
    08224252996342955972118994246 3 Alice sends to Bob x = g**a%p
y = 2105536764708523218638657110655551339688586778013424465038540 5
    9775644346276419307309962854893414483737776066061285480650119 0
    81326357437578023680749360471 2 Bob sends to Alice y = g**b%p

key_A = 3757814157424326770605057448102820498331116479265196100321
        5901263711170130759929166621710543994302222852279832056640
        6576388874225161692884245970918228315 4 shared key on Alice's side y**a%p
key_B = 3757814157424326770605057448102820498331116479265196100321
        5901263711170130759929166621710543994302222852279832056640
        6576388874225161692884245970918228315 4 shared key on Bob's side x**b%p
```

Diffie Hellman – Final Remarks

- Recall that the length of the prime p in bits is $n = \lfloor \log_2 p \rfloor + 1$.
- Computation time for exchanging the key is $O((\log_2 p)^3) = O(n^3)$ bit operations.

- DH key exchange is at most as secure as discrete log.
- Formal equivalence between DH (Diffie-Hellman key distribution) and DL (discrete logarithm problem) has never been proved, though some partial results are known.
- Over the last 36 years there were many attempts to crack the scheme. None succeeded, and DH key exchange (with an appropriately large prime p, *e.g.* 1024 bits) is considered secure.

- U.S. Patent 4,200,770, now expired, describes the algorithm and credits Hellman, Diffie, and Merkle as inventors, and the three of them have joined the Hall of Fame.