Modern Cryptology - Problem Set 1. name: Guy Levy. ID: 206865362.

1: GCD

1.1 Facts about gcd

1. $gcd(an+b,n) = gcd(b,n)$:

   $gcd(an+b,n) = max\{x : x|an+b \wedge x|n\} =^{(*)} max\{x : x|b \wedge x|n\} = gcd(b,n)$
   explanation of (*): we shall prove: $x|an+b \wedge x|n \iff x|b \wedge x|n$:

   $(\Rightarrow)$ $x|an+b \wedge x|n$
   $\Rightarrow$ there exists $c, d \in \mathbb{Z}$ such that $cx = an+b \wedge dx = n$
   $\Rightarrow cx = adx + b$
   $\Rightarrow (c-ad)x = b$
   $\Rightarrow x|b$
   $\Rightarrow x|b \wedge x|n$

   $(\Leftarrow)$ $x|b \wedge x|n$
   $\Rightarrow$ there exists $c, d \in \mathbb{Z}$ such that $cx = b \wedge dx = n$
   $\Rightarrow$ for any $a \in \mathbb{Z} : an+b = adx + cx = (ad+c)x$
   $\Rightarrow x|an+b \wedge x|b$

2. $gcd(a,n)|gcd(ab,n)$:
   An equivalent way of defining $gcd(ab,n)$ is the number in the set $\{ x : x|ab \wedge x|n \}$
   such that every other member d of this set divides it.
   All common divisors of ab, n divide gcd(ab,n).
   $gcd(a,n)$ is a divisor of ab, n, so as such it divides their gcd and so we get what we wanted to prove.

3. Let $p \in \mathbb{Z}$ be a prime number. If $p|ab$ then $p|a$ or $p|b$:
   a, b can be represented as products of primes.
   So we denote:
   $a = \Pi p_i^{k_i}, b = \Pi p_i^{m_i}, \Rightarrow ab = \Pi p_i^{k_i+m_i}$.
   Assume p is the jth prime.
   From $p|ab$ it folllows $k_j + m_j > 1$.
   Remember $\forall i \in \mathbb{N} : k_i, m_i \in \mathbb{N}$ so it follows that $m_j > 1 \vee k_j > 1$.
   So $p|a \vee p|b$.

4. $gcd(a,n) = gcd(b,n) = 1 \Rightarrow gcd(ab,n) = 1$:
   Denote $g_1 = gcd(a,n), g_2 = gcd(b,n), g_3 = gcd(ab,n)$.
   Assume by contradiction $g_3 > 1$.
   $g_3|ab \wedge g_3|n$.
   $\Rightarrow$ There exists a prime p such that $p|g_3$ so $p|ab$ and $p|n$.
   $\Rightarrow (p|a \vee p|b) \wedge p|n$.
   $\Rightarrow (p|a \wedge p|n) \vee (p|b \wedge p|n)$.
   $\Rightarrow p|g_1 \vee p|g_2$.
   $\Rightarrow g_1 > 1 \vee g_2 > 1$. And thats a contradiction.

1.2 extended euclidean algorithm

In [1]:

```python
def gcd(a,b):
    """     given a,b computes their gcd and x,y such that a*x + b*y = gcd.
    returns those in the form of a tuple (gcd,x,y)    """
    if a == b:
        return (a,1,0)
    elif a < b:
        a,b = b,a
    a0,b0 = a,b
    if b == 0:
        return (a,1,0)
    if a % b == 0:
        return (b,0,1)

    As, Bs = [a], [b]
    Rs, Ds = [], [] # remainders, multipliers

    lastR = None
    while lastR != 0:     # a = d*b + r ==> (a,b) = (b,r).
        Rs.append(a % b) ; Ds.append(a // b) # saving remainders and multipliers.
        lastR = Rs[-1]
        As.append(b)     ; Bs.append(lastR)
        a, b = As[-1], Bs[-1]
    gcd = As[-1]

    # from the euclidian process we got equations and now we use them to compute x,y
    Ds.pop()
    y, x = 1, -Ds.pop()
    for i in range(len(Ds)):
        if i%2 == 0:
            y = y - x*Ds[-1-i]
        else:
            x = x - y*Ds[-1-i]
    if a0*x + b0*y == gcd:
        return (gcd,x,y)
    return (gcd,y,x)
```

time complexity of worst case is $log(N)$ where $N = max(a, b)$

1.3 computing inverses in $\mathbb{Z}_n^*$

Algorithm: Given $a \in \mathbb{Z}_n^*$ we compute the Extended Euclidean Algorithm on (a, n), we get some result (gcd, x, y) and return x.

Correctness: We know by definition of $\mathbb{Z}_n^*$ that $gcd(a, n) = 1$ so by applying the Extended Euclidean Algorithm on (a, n) we get result $(1, x, y)$: $x, y \in \mathbb{Z}$ such that:

$ax + ny = 1$

$\Rightarrow ax(mod n) = 1$

$\Rightarrow$ x is the inverse of a by definition.

1.4 $\mathbb{Z}_n^*$ is a group

Closure: let there be $a, b \in \mathbb{Z}_n^*$.
$gcd(a,n) = gcd(b,n) = 1 \Rightarrow^{(1.1:4)} gcd(ab,n) = 1 \Rightarrow^{(1.1:1)} gcd(ab(modn),n) = 1$
also $0 \leq ab(modn) < n \Rightarrow ab(modn) \in \mathbb{Z}_n^*$

Associativity: let there be $a, b, c \in \mathbb{Z}_n^*$.
$((a*b)(modn)*c)(modn) = ((a*b)*c)(modn) = (a*(b*c))(modn) = (a*(b*c)(modn))(modn)$

Identity: For all $a \in \mathbb{Z}_n^* : 1 * a(modn) = a$ so 1 is the identity.

Inverse: Follows immediatly from the correctess proof of the algorithm in (1.3).

2: Modular Exponentiation

2.1 general groups

I suggest the following algorithm. given g,x we return with recursion the result of
$alg(g,x) = alg(g^2, \lfloor x/2 \rfloor) * g^{x(mod2)}$.
with stoping conditions of $g^1 = g$ and $g^0 = 1$.
complexity: every recursive call is doing at most 1 group operation plus some O(1) calculations. there will be at most log(x) calls.
so overall the complexity is $O((log(x)+c)*log(|G|)) = O(log(x)*log(|G|))$.

In [2]:

```python
class Group: # ============= not important ============
    def __init__(self, op):
        self.identity = 1
        self.op = op
    def operation(self,a,b):
        return self.op(a,b)
G = Group(lambda a,b: a*b)
```

In [3]:

```python
def g_exp_x(G,g,x): # ========= the algorithm =============
    if x == 0:
        return G.identity
    if x == 1:
        return g
    return G.operation( g_exp_x(G, G.operation(g,g), x//2), g if x%2 else G.identity )
```

2.2 over $\mathbb{Z}_n^*$

The previous algorithm does exactly what is needed, further,
I will prove the same algorithm achieves the required time complexity.
Time complexity of the algorithm for general group is $O((\text{operation-time}) * log(|G|))$.
Now because $|\mathbb{Z}_N^*| = N$ the time complexity we get is
$O(poly(log(x), log(N)) * log(N)) = O(poly(log(x), log(N)))$

In [4]:

```
N = 25
Z_N = Group(lambda a,b: (a*b)%N)
print(g_exp_x(Z_N, 23, 3)) # 23^3(mod 25) == 17
```

```
17
```

3: Data Processing Inequality

3.1 Statistical Distance

Assume, for the sake of contradiction that: $SD(f(X), f(Y)) > SD(X, Y)$.

Remember : $SD(X, Y) = max_D |adv_D(X, Y)|$.

Let us build D' such that $adv_{D'}(X, Y) > SD(X, Y)$ and by that reach a contradiction to the maximality of $SD(X, Y)$.

First lets define another distinguisher D'' :
D'' is the distinguisher such that $adv_{D''}(f(X), f(Y)) = SD(f(X), f(Y))$.(exists from definition of SD)

D' is the distinguisher that takes distributions X, Y, applies f on them and computes D''.

By definition of D' :

$$adv_{D'}(X, Y) = adv_{D''}(f(X), f(Y)) = SD(f(X), f(Y))$$

That is a contradiction because:

$$adv_{D'}(X, Y) = SD(f(X), f(Y)) > SD(X, Y) = max_D |adv_D(X, Y)|$$

3.2 Computational Distance

$X \approx^c Y$.
$\Rightarrow \forall$ PPT $D : |Pr_{x \leftarrow X_n}[D(x) = 1] - Pr_{x \leftarrow Y_n}[D(x) = 1]| \leq negl(n)$.
In particular for any D' that computes f on the input (which takes polynomial time) and then computes some PPT D'' on the result, it holds that D' is PPT.
So it follows:

$$\forall \text{ PPT } D' : |Pr_{x \leftarrow X_n}[D'(x) = 1] - Pr_{x \leftarrow Y_n}[D'(x) = 1]| \leq negl(n)$$

$\Rightarrow^{(D'' \text{defines} D')}$

$$\forall \text{ PPT } D'' : |Pr_{x \leftarrow X_n}[D'(x) = 1] - Pr_{x \leftarrow Y_n}[D'(x) = 1]| \leq negl(n)$$

$\Rightarrow$

$$\forall \text{ PPT } D'' : |Pr_{x \leftarrow X_n}[D''(f(x)) = 1] - Pr_{x \leftarrow Y_n}[D''(f(x)) = 1]| \leq negl(n)$$

$\Rightarrow$

$$\forall \text{ PPT } D'' : |Pr_{x \leftarrow f(X_n)}[D''(x) = 1] - Pr_{x \leftarrow f(Y_n)}[D''(x) = 1]| \leq negl(n)$$

$\Rightarrow$

$$f(X) \approx^c f(Y)$$

4: Semantic Security for Perfect Encryption

Let there be adversary A.
Assume, for the sake of contradiction that there is no such "simulator" S.

$\Rightarrow \exists m_1, m_2 : A(C_1) \nsim^{dist} A(C_2)$ when C_i denotes $E_K(m_i)$
(I will use the sign $\sim^{dist}$ to denote that two random variables have the same distribution)
That is because if there was no pair m_1, m_2 such as that then for all C's the $A(C)$'s distributes exactly the same, so an S with that fixed distribution exists.

Because $A(C_1) \nsim^{dist} A(C_2)$ there exists m such that: $Pr[A(C_1) = m] \neq Pr[A(C_2) = m]$

Let D_m be a distinguisher that given c answers 1 if $A(c) = m$. else it answers 0.

Then: $|Pr_{c \leftarrow C_1}[D_m(c) = 1] - Pr_{c \leftarrow C_2}[D_m(c) = 1]| > 0$

$\Rightarrow C_1 \nsim^{dist} C_2$

Which implies directly (using lemma from class) that (G, E, D) is not perfectly secure. Thats a contradiction.

5: Code Breaking

5.1 Crack the Code

plaintext:

talking to her he realized how easy it was to present an appearance of orthodoxy while having no grasp whatever of what orthodoxy meant in a way the world view of the party imposed itself most successfully on people incapable of understanding it they could be made to accept the most flagrant violations of reality because they never fully grasped the enormity of what was demanded of them and were not sufficiently interested in public events to notice what was happening by lack of understanding they remained sane they simply swallowed everything and what they swallowed did them no harm because it left no residue behind just as a grain of corn will pass un digested through the body of a bird as a bonus the first student to send the name of the book from which this paragraph was taken to shafik will get a bonus

5.2 Explain the Algorithm

the plaintext was split to 4 sections of equal length, each section used a mono-alphabetic substitution cipher (permutation of letters) the permutations used from (plaintext to cipher) correspond to the following python dictionaries:

In [5]:

```
# permutation for the 1st section
key1 = {'t': 'i', 'a': 'v', 'l': 't', 'k': 'g', 'i': 'u', 'n': 'm', 'g': 'f',
    'o': 'z','h': 'r', 'e': 'a', 'r': 'c', 'z': 'e', 'd': 's', 'w': 'h','s': 'b',
    'y': 'y','p': 'j', 'c': 'w', 'f': 'x', 'b': 'p', 'v': 'k', 'm': 'd', 'u': 'q'}
# permutation for the 2nd section
key2 = {'y': 'i', 'o': 'h', 'n': 'a', 'p': 'w', 'e': 'd', 'l': 'r', 'i': 'v',
        'c': 'e', 'a': 'o', 'b': 'g', 'f': 'm', 'u': 'f', 'd': 'p', 'r': 's',
        's': 't', 't': 'q', 'g': 'k', 'h': 'x', 'm': 'b', 'v': 'c', 'w': 'y'}
# permutation for the 3rd section
key3 = {'i': 's', 'c': 'j', 'e': 't', 'n': 'w', 't': 'e', 'l': 'l', 'y': 'h',
        'r': 'q', 's': 'f', 'd': 'i', 'p': 'y', 'u': 'm', 'b': 'o', 'v': 'r',
        'o': 'b', 'w': 'p', 'h': 'g', 'a': 'k', 'g': 'c', 'k': 'a', 'f': 'u', 'm': 'n'}
# permutation for the 4th section
key4 = {'n': 'i', 'o': 'p', 'r': 'y', 'e': 's', 's': 'd', 'i': 'x', 'd': 'k',
        'u': 'z', 'b': 'n', 'h': 'o', 'j': 'm', 't': 'v', 'a': 'w', 'g': 'u',
        'f': 'h', 'c': 'r', 'w': 'f', 'l': 'c', 'p': 't', 'y': 'j', 'm': 'b', 'k': 'l'}
```

These can be specified using a key of about 88*4 = 352 bits ($26! \approx 2^{88}$).