Cryptology I (spring 2019)	Dominique Unruh

Exercise Sheet 3

Out: 2019-04-11 Due: 2019-04-25

Problem 1: Textbook RSA and hybrid encryption

A common variant of textbook RSA is the following: During key generation, the modulus N is chosen as usual. We chose e as $e := 3$ (instead of random). Then d is chosen with $ed \equiv 1 \bmod \varphi(N)$ (as usual). This is implemented by the Python functions `rsa_keygen`, `rsa_enc`, `rsa_dec` below.

We use this in a "hybrid encryption", which first picks an AES key k, encrypts it with RSA, and then encrypts the actual message with AES using the key k. (Functions `hyb_enc`, `hyb_dec`.)

Your task is to write an adversary that, given the public key `pk`, and the hybrid encryption `c` of some message `m`, finds `m`. That is, fill in the function body of the function `adv` below so that the function `test_adv` prints `Success. The adversary broke the scheme.`

Hint: Note that $k < \sqrt[3]{N}$ here. This makes it much simpler to find k given $k^3 \bmod N$!

(You find the following file on the lecture webpage, too.)

```
#!/usr/bin/python3

# Use "pip install sympy" (possibly with sudo) to install sympy
# And "Crypto" might need "pip install pycrypto" if it's not installed

import sympy, math, Crypto, random

prime_len = 1024

# Copied from http://stackoverflow.com/questions/4798654/modular-multiplicative-inverse-fu
def egcd(a, b):
    if a == 0:
        return (b, 0, 1)
    else:
        g, y, x = egcd(b % a, a)
        return (g, x - (b // a) * y, y)
def modinv(a, m):
    g, x, y = egcd(a, m)
    if g != 1:
        raise Exception('modular inverse does not exist')
    else:
```

```
        return x % m

def rsa_keygen():
    while True:
        try:
            p = sympy.ntheory.generate.randprime(2**prime_len,2**(prime_len+1))
            q = sympy.ntheory.generate.randprime(2**prime_len,2**(prime_len+1))
            e = 3
            N = p*q
            phiN = (p-1)*(q-1)
            pk=(N,e)
            sk=(N,modinv(e,phiN))
            return (pk,sk)
        except Exception as e:
            pass

# Rough ad-hoc algorithm, not optimized
def exp_mod(a,e,N):
    res = 1
    b = a
    i = 0
    while e>=2**i: # Invariant: b=a**(2**i)
        if e & 2**i != 0:
            e -= 2**i
            res = (res*b) % N
        b=(b*b) % N
        i += 1
    assert e==0
    return res

# Just a test
assert exp_mod(23123,323,657238293) == ((23123**323) % 657238293)

def rsa_enc(pk,m):
    (N,e) = pk
    return exp_mod(m,e,N)

def rsa_dec(sk,c):
    (N,d) = sk
    return exp_mod(c,d,N)

def int_to_bytes(i,len): # Not optimized
    res = []
```

```
    for j in range(len):
        res.append(i%256)
        i = i>>8
    return bytes(res)

def aes_cbc_enc(k,m):
    from Crypto.Cipher import AES
    from Crypto import Random
    assert len(m)%AES.block_size == 0
    k = int_to_bytes(k,AES.block_size)
    iv = Random.new().read(AES.block_size)
    cipher = AES.new(k, AES.MODE_CBC, iv)
    return iv + cipher.encrypt(m)

def aes_cbc_dec(k,m):
    from Crypto.Cipher import AES
    from Crypto import Random
    k = int_to_bytes(k,AES.block_size)
    iv = m[:AES.block_size]
    cipher = AES.new(k, AES.MODE_CBC, iv)
    return cipher.decrypt(m[AES.block_size:])

# Just a test
assert aes_cbc_dec(2123414234,aes_cbc_enc(2123414234,b'hello there test')) == b'hello ther

def hyb_enc(pk,m):
    assert isinstance(m,bytes)
    k = random.getrandbits(256)
    aes_k_m = aes_cbc_enc(k,m)
    assert m == aes_cbc_dec(k,aes_k_m)
    rsa_pk_k = rsa_enc(pk,k)
    return (rsa_pk_k,aes_k_m)

def hyb_dec(sk,c):
    (c1,c2) = c
    k = rsa_dec(sk,c1)
    m = aes_cbc_dec(k,c2)
    return m

def adv(pk,c):
    m = b"put the right message here"
    return m
```

```
def test_adv():
    (pk,sk) = rsa_keygen()
    # Generate a message m
    m = b"a few random words to be shuffle randomly to get some interesting ciphertext not
    random.shuffle(m)
    m = b" ".join(m)
    # Get a key pair
    (pk,sk) = rsa_keygen()
    # Encrypt m
    c = hyb_enc(pk,m)
    # Just a test
    assert m == hyb_dec(sk,c)
    # Call the adversary, let him guess m
    m2 = adv(pk,c)
    assert isinstance(m2,bytes)
    # Check
    if m==m2:
        print("Success. The adversary broke the scheme")
    else:
        print("*** Failure ***")

test_adv()
```

Problem 2: Malleability of textbook RSA

The adversary get a textbook RSA encryption $c = E(pk, m)$ for some unknown message m. The adversary also knows $pk = (N, e)$. The adversary wants to compute $c' = E(pk, 2m)$. (This is a specific example of malleability.) How can the adversary efficiently compute c' from c and pk?

You may assume that $0 \leq m < N/2$.

Problem 3: Encoding messages for ElGamal (bonus problem)

The message space of ElGamal (when using the instantation that operates modulo a prime $p > 2$ with $p \equiv 3 \bmod 4$ [1]) is the set $\mathrm{QR}_p = \{x^2 \bmod p : x = 0, \ldots, p-1\}$.

The problem is now: if we wish to encrypt a message $m \in \{0,1\}^\ell$ (with $\ell \leq |p| - 2$), how do we interpret m as an element of QR_p?

One possibility is to use the following function $f : \{1, \ldots, \frac{p-1}{2}\} \to \mathrm{QR}_p$:

$$f(x) := \begin{cases} x & \text{if } x \in \mathrm{QR}_p \\ -x \bmod p & \text{if } x \notin \mathrm{QR}_p \end{cases}$$

[1] You do not actually need to use this fact, but the hint that $-1 \notin \mathrm{QR}_p$ below is only true in this case.

Once we see that f is a bijection and can be efficiently inverted, the problem is solved, because a bitstring $m \in \{0,1\}^\ell$ can be interpreted as a number in the range $1, \ldots, \frac{p-1}{2}$ by simply interpreting m as a binary integer and adding 1 to it. (I.e., we encrypt $f(m+1)$.)

We claim that the following function is the inverse of f:

$$g(x) := \begin{cases} x & \text{if } x = 1, \ldots, \frac{p-1}{2} \\ -x \bmod p & \text{if } x \neq 1, \ldots, \frac{p-1}{2} \end{cases}$$

We thus need to show the following: the range of f is indeed QR_p, and that $g(f(x)) = x$ for all $x \in \{1, \ldots, \frac{p-1}{2}\}$.

(a) Show that $f(x) \in \mathrm{QR}_p$ for all $x \in \{1, \ldots, \frac{p-1}{2}\}$.

Hint: You can use (without proof) that $-1 \notin \mathrm{QR}_p$ (this only holds in QR_p for p prime with $p \equiv 3 \bmod 4$). And that the product of two quadratic non-residues is a quadratic residue (this only holds in QR_p, but not in QR_n for n non-prime).

(b) Show that $g(f(x)) = x$ for all $x \in \{1, \ldots, \frac{p-1}{2}\}$.

(This then shows that f is injective and efficiently invertible. Bijectivity follows from injectivity because the domain and range of f both have the same size.)

Hint: Make a case distinction between $x \in \mathrm{QR}_p$ and $x \notin \mathrm{QR}_p$. Show that for $x \in \{1, \ldots, \frac{p-1}{2}\}$ it holds that $-x \bmod p \notin \{1, \ldots, \frac{p-1}{2}\}$.

Problem 4: Malleability of ElGamal

Remember the auction example from the lecture: Bidder 1 produces a ciphertext $c = E(pk, bid_1)$ where E is the ElGamal encryption algorithm (using integers mod p as the underlying group). Given c, Bidder 2 can then compute c' such that c' decrypts to $2 \cdot bid_1 \bmod p$. This allows Bidder 2 to consistently bid twice as much as Bidder 1.[2]

Now refine the attack. You may assume that bid_1 is the amount of Cents Bidder 1 is willing to pay. And you can assume that Bidder 1 will always bid a whole number of Euros. (I.e., bid_1 is a multiple of 100.)

Show how Bidder 2 can consistently overbid Bidder 1 by only 1 %. What happens to your attack if Bidder 1 suddenly does not bid a whole number of Euros?

Hint: Remember that modulo p, one can efficiently find inverses. For example, one can find a number a such that $a \cdot 100 \equiv 1 \bmod p$.

[2] As long as $bid_1 < p/2$, that is. Otherwise $2 \cdot bid_1 \bmod p$ will not be twice as much as bid_1. However, for large p, $bid_1 \geq p/2$ is an unrealistically high bid.

Problem 5: Hybrid encryption – implementations

(a) Implement a hybrid encryption using ElGamal and AES. You are allowed to use ready-made ElGamal and AES.

In the contributed file `hybrid.py` (lecture webpage), you find a prepared template in Python that already provides function for ElGamal and AES encryption as well as some utility functions and testing code that you might need. I recommend to use that code. If you wish to use another language, you will have to find your own ElGamal and AES routines.

You should check that `hybrid_decrypt(sk,hybrid_encrypt(pk,msg))` returns `msg`.

It is OK if you only allow encrypting messages whose length is a multiple of 16 bytes (blocklength of AES).

(b) **[Bonus problem]** The ElGamal implementation used in `hybrid.py` might leak whether the message `msg` is a quadratic residue. Using the methods developed in Problem 3, fix the functions `elgamal_encrypt` and `elgamal_decrypt` to avoid this leakage. (You need to make sure that `elgamal_decrypt(sk,elgamal_encrypt(pk,msg))` still returns `msg`.)

Problem 6: Security proofs (bonus problem)

Recall the definition of IND-OT-CPA (Definition 3 in the lecture notes). There, we defined security by saying that if the adversary cannot distinguish between an encryption of m_0 and m_1 in the sense that it will output 1 will almost the same probability in both cases.

Consider the following variant of the definition:

Definition 1 (IND-OT-CPA – variant) *An encryption scheme (KG, E, D) is (τ, ε)-IND-OT-CPA' secure*[3] *if for any τ-time algorithm A we have that*

$$\left|\Pr[b^* = b : b \xleftarrow{\$} \{0,1\},\ k \leftarrow KG(),\ (m_0, m_1) \leftarrow A(),\ c \leftarrow E(k, m_b),\ b^* \leftarrow A(c)] - \frac{1}{2}\right| \leq \varepsilon.$$

(Here we quantify only over algorithms A that output (m_0, m_1) with $|m_0| = |m_1|$.)

That is, the message m_b is encrypted (with b random), and we want that the adversary cannot guess b with probability much different from $\frac{1}{2}$. (Guessing with probability $\frac{1}{2}$ is always possible since b is just a single bit.)

We wish to prove that if (KG, E, D) is (τ, ε)-IND-OT-CPA' secure, then (KG, E, D) is $(\tau, 2\varepsilon)$-IND-OT-CPA secure.

Note: The converse also holds, but we will not prove that.

[3] This is not an established name!

(a) Assume an adversary A that breaks $(\tau, 2\varepsilon)$-IND-OT-CPA security. Let

$$\alpha_0 := \Pr[b^* = 1 : k \leftarrow KG(), (m_0, m_1) \leftarrow A(), c \leftarrow E(k, m_0), b^* \leftarrow A(c)]$$

and

$$\alpha_1 := \Pr[b^* = 1 : k \leftarrow KG(), (m_0, m_1) \leftarrow A(), c \leftarrow E(k, m_1), b^* \leftarrow A(c)].$$

What do we know about α_0 and α_1 (by definition of IND-OT-CPA and the fact that A breaks IND-OT-CPA)?

(b) Compute

$$\beta := \Big|\Pr[b' = b : k \leftarrow KG(), b \xleftarrow{\$} \{0,1\}, (m_0, m_1) \leftarrow A(), \\ c \leftarrow E(k, m_b), b' \leftarrow A(c)] - \tfrac{1}{2}\Big|.$$

(As a formula using α_0 and α_1.)

(c) Using (a) and (b), show that if A breaks $(\tau, 2\varepsilon)$-IND-OT-CPA, then A breaks (τ, ε)-IND-OT-CPA'. (Hence: IND-OT-CPA' implies IND-OT-CPA.)