Cryptology I (spring 2018) Dominique Unruh

Exercise Sheet 4

Out: 2018-03-15 Due: 2018-03-23

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: We discuss/discussed in the practice the problem with RSA with $e = 3$ when RSA-encrypting short messages.

(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: 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[1] *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.

(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

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

(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.)

[1] This is not an established name!