Cryptology I (spring 2015) Dominique Unruh

Exercise Sheet 3

Out: 2015-03-26 Due: 2015-04-08

Problem 1: 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.[1]

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 $a \cdot 100 \equiv 1 \bmod p$.

Problem 2: 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 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.)

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

[1] 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.

```
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 = b""
    for j in range(len):
        res += chr(i%256)
        i = i>>8
    return 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,'hello there test')) == 'hello there

def hyb_enc(pk,m):
    k = random.getrandbits(256)
```

```
    aes_k_m = aes_cbc_enc(k,m)
    assert aes_cbc_dec(2123414234,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 = "put the right message here"
    return m

def test_adv():
    (pk,sk) = rsa_keygen()
    # Generate a message m
    m = "a few random words to be shuffle randomly to get some interesting ciphertext not
    random.shuffle(m)
    m = " ".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)
    # Check
    if m==m2:
        print "Success. The adversary broke the scheme"
    else:
        print "*** Failure ***"

test_adv()
```