CS 427/519: Homework 7

Due: **Wednesday** March 14, 10pm; **typed** and **submitted electronically**.

1. In the data file on the website I have provided:

   - Parameters for ElGamal encryption (prime p, primitive root g)
   - An ElGamal private key pair (a)
   - An ElGamal ciphertext (B, C)

   Please tell me:

   (a) What is the public key? How did you compute it?

   (b) What is the plaintext? How did you compute it?

2. Alice and Bob are going to perform three instances of Diffie-Hellman key agreement. The protocol says that each time they should choose a, b exponents randomly. Unfortunately, the programmer that they hired was asleep during the DHKA lecture in CS427, and provided a stupid implementation of DHKA. Instead of random exponents, they use sequential exponents. That is, Alice & Bob use random exponents a, b in the first DHKA, then $a+1, b+1$ in the second, $a+2, b+2$ in the third.

   Eve observes all of these DHKA interactions as an eavesdropper. She later learns the key derived from the **second** DHKA instance. Describe how she can now compute the first and third DHKA keys. Then carry out the attack on the concrete values provided in the data file.

3. Recall "textbook RSA" signatures:

```
KeyGen:
    choose primes $p, q$
    $N := pq$
    choose $e, d$ with $ed \equiv_{\phi(N)} 1$
    $sk := (N, d)$
    $vk := (N, e)$
    return $(vk, sk)$

Sign$\Big((N, d), m\Big)$:
    return $m^d \bmod N$

Ver$\Big((N, e), m, \sigma\Big)$:
    return $\sigma^e \overset{?}{\equiv}_N m$
```

   These were insecure because an attacker can simply choose σ first, then solve for $m \equiv_N \sigma^e$. Now σ is a valid signature on m, hence a forgery.

   Rabin signatures taught us that computing square roots mod N is a hard problem. Let's try to use that fact by making the following change to textbook RSA:

```
KeyGen:
    choose primes $p, q$
    $N := pq$
    choose $e, d$ with $ed \equiv_{\phi(N)} 1$
    $sk := (N, d)$
    $vk := (N, e)$
    return $(vk, sk)$

Sign$\Big((N, d), m\Big)$:
    return $(m^2)^d \bmod N$

Ver$\Big((N, e), m, \sigma\Big)$:
    return $\sigma^e \overset{?}{\equiv}_N m^2$
```

This modification successfully thwarts the previous attack. An attacker can choose σ and then compute $x \equiv_N \sigma^e$. But instead of presenting (x, σ) as a forgery like before, now the attacker must present σ and a value m such that $m^2 \equiv_N x$. In other words, the attacker must compute a square root mod N, which is hard!

Despite all of these good intentions, show that the new scheme is still not a secure signature scheme. Formally describe your attack (calling program + bias). Libraries for security of signatures are given below.

Hint: Ask for one *or more* signatures of chosen messages. Then compute the signature of a *related* message.

grad. Describe a randomized algorithm RefreshRand that takes as input an ElGamal public-key pk + ciphertext c, and outputs a new ciphertext, with the following property:

- Suppose c is an ElGamal encryption of some (unknown) plaintext m. Then RefreshRand(pk, c) and Enc(pk, m) should be *identically distributed.*

Note that RefreshRand does **not** know m. So, without knowing m it is possible to generate an encryption of the same message, which is distributed just as a "fresh" encryption (and hence does not look like it was derived from the original ciphertext).

Please describe the algorithm and argue/prove why it has this property.

Security of Signatures:

```
$\mathcal{L}^{\Sigma}_{\text{sig-real}}$
$(vk, sk) \leftarrow \Sigma.\text{KeyGen}$

GETVK():
    return $vk$

GETSIG($m$):
    return $\Sigma.\text{Sign}(sk, m)$

VER($m, \sigma$):
    return $\Sigma.\text{Ver}(vk, m, \sigma)$
```

$\approx$

```
$\mathcal{L}^{\Sigma}_{\text{sig-fake}}$
$(vk, sk) \leftarrow \Sigma.\text{KeyGen}$
$\mathcal{S} := \emptyset$

GETVK():
    return $vk$

GETSIG($m$):
    $\sigma := \Sigma.\text{Sign}(sk, m)$
    $\mathcal{S} := \mathcal{S} \cup \{(m, \sigma)\}$
    return $\sigma$

VER($m, \sigma$):
    return $(m, \sigma) \overset{?}{\in} \mathcal{S}$
```