Andrew Johnson

CS 427

HW6

1. 410222175196
2. (a) Assuming we know the public key *A* and the group generator *g*. Given an ElGamal encryption of *M, (B,C).* Thus we have *A, g, B,* and *C*.

|  |
| --- |
| DEC(a,(B,C)) |
| K = Ba |
| M = CK-1 |
| return M |

We need to find a *B’* and *C’* such that when sent into the decryption we still get back *M*.

Let *B’ = g ∙ B* and *C’ = A ∙ C*. When sent into the decryption we see that

*K’ = (B’)a = (gB)a = ga ∙ Ba = A ∙ K*

Then *M’= C’ ∙ (K’)-1 = (A ∙ C) ∙ (A ∙ K)-1 = A ∙ A-1 ∙ C ∙ K-1 = C ∙ K-1 = M* □

(b) Given two ElGamal encryptions of *M1* and *M­2*, we get *(B1,C1)* and (B2,C­­2). We need to find a B’ and C’ such that when decrypted we get *M1 ∙ M2*.

Let *B’ = B1 ∙ B­2* and *C’ = C1 ∙ C2*. When these are put into the decryption we see that

*K’ = (B1 ∙ B­2)a = B1a ∙ B­2a = K1 ∙ K2*

*M’ = (C­1 ∙ C2) ∙ (K’)-1 = (C­1 ∙ C2) ∙ (K1 ∙ K2)-1 = (C1 ∙ K1-1) ∙ (C2 ∙ K2-1) = M1* ∙ M2 □

1. (a) If we take the case where x = 0 then we see that it is the same as the Lemma stated before. So what we are really checking is if *ri – rj ≡ x (mod p)*. So we are checking the distance between each r. In the former lemma we were checking a distance of 0. In the new Lemma, we can take any arbitrary x and will have the same probability instead of only looking at 0.

(b) Assume we know integers r and s, such that *gr ≡ X ∙ gs (mod p)*, We already know that g is a primitive root of *Z\*p*, so we know that any power of *g* must have an inverse. Then we can use *g-s* on our assumption to get, *gr ∙ g-s ≡ X ∙ (gs ∙ g-s) (mod p)* *→ gr - s ≡ X (mod p).* We can say *x = r – s* and then we have proved *gx ≡ X (mod p).* □

(c) Let *g* be a primitive root of *Z\*p­­*. From 3(a) we know with a *.6* probability that we can find an *r* and *s* that exist in *Z\*p­­* such that *r – s = x*, for any fixed x, by only taking elements of *g*. This means we can find an *r* and *s* that satisfy above in, or simplified. Thus, we see that by finding *r* and *s*, we have also satisfied 3(b) which is for finding the discrete logarithm. With both of these we can find *x* to solve the discrete logarithm in time.