# Answers 1.

## a)

p(C=2) = p(P=a)p(K=k5) + p(P=c)p(K=k1) + p(P=d)p(K=k2) + p(P=d)p(K=k4) + p(P=e)p(K=k3)

= 0.2 \* (p(P=a) + p(P=c) + p(P=d) + p(P=d) + p(P=e))

= 0.2 \* (0.05 + 0.21 + 0.17 + 0.17 + 0.33)

= 0.186

p(C=4) = p(P=b)p(K=k3) + p(P=b)p(K=k4) + p(P=c)p(K=k2) + p(P=c)p(K=k5) + p(P=e)p(K=k1)

= 0.2 \* (p(P=b) + p(P=b) + p(P=c) + p(P=c) + p(P=e))

= 0.2 \* (0.24 + 0.24 + 0.21 + 0.21 + 0.33)

= 0.246

## b)

p(C=2|P=b) = 0 as b cannot encode 2 by the table

p(C=4|P=b) = 0.2 + 0.2

p(P=b|C=4) = p(C=4|P=b) p(P=b) / p(C=4) = 0.4 \* 0.24 / 0.246 = 0.390

## c) i.

The same plaintext encodes to the same ciphertext for different keys. So the mathematical formulation of perfect secrecy doesn’t hold i.e.

p(P=b|C=4) = 0.390 =/= 0.24 = p(P=b)

## c) ii.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | a | b | c | d | e |
| k1 | 4 | 2 | 1 | 3 | 5 |
| k2 | 5 | 1 | 4 | 2 | 3 |
| k3 | 1 | 3 | 2 | 5 | 4 |
| k4 | 3 | 4 | 5 | 1 | 2 |
| k5 | 2 | 5 | 3 | 4 | 1 |

(copied from 15-16)

All rows and columns have 1,2...5 (and no more). Thus there is no overlap, and the ciphertext looks truly random to an attacker. Therefore p(P=m|C=c) = p(P=m) holds for all.

## d) i.

The key stream Y=Y1Y2Y3 ... Yn is first generated (purely from the initialisation vector IV), which is as long as the message. Then the ciphertext is calculated by taking the XOR of this key stream with the message; c = m 𐌈 Y . So this ciphertext is calculated just the same as OTP.

However, since the key stream is generated purely from the initialisation vector, it only has as many possibilities as this IV. This limits the key space, so it’s not equivalent to OTP, and is not perfectly secure.

## d) ii.

Since m 𐌈 m’ = 100.... the first bits will be the bit flip of each other, otherwise they’re the same. Then since c = m 𐌈 Y, and the key streams will be the same, again, the first bits will be the bit flip of each other, otherwise they’re the same.

So they only differ in the first bit which are bitflips.

# 2.

## a) i.

1 has order 1, as 1s = 1. p-1 = -1 has order 2, as (-1)s = 1. Multiplication in groups of order 1, 2 is really simple, especially, simple to reverse, so our assumptions about the discrete logarithm being hard are invalid. So the system would not be secure.

## a) ii.

The size of the multiplicative group is p-1. So by Fermat’s Little/Euler’s Theorem 𝛼p-1 = 1 (mod p), so 1 will be sent, which is an obvious special case which can be reversed.

## b) i.

We’ve assumed p = Rq + 1. ℤ\*p has order p-1 = Rq.

By Lagrange’s theorem, the orders of subgroups are the divisors: 1, q, R (or the divisors of R if it’s not prime).

If 𝛼 was the generator of a subgroup of order q, 𝛼q would now be 1 (by Fermat’s Little/Euler’s Theorem).

So it’s either 1 or a generator of a group of size R i.e. order R.

Not sure how we rule out 1 here? But in practise if we got 1, we’d chuck it out and repeat, so I think this is ok?

Better answer I think:

Let’s show that βR+1 = β, which means that β only generates R elements in ℤ\*p

βR+1 = 𝛼(R+1)q mod p = 𝛼Rq 𝛼q mod p = 𝛼q mod p = β mod p

Because 𝛼Rq = 1 mod p using Fermat’s Theorem (Rq + 1 = p is a prime and do not divide 𝛼)

## b) ii.

Assuming Eve can MITM. Eve can replace 𝛼x by (𝛼q)x , then as well, replace 𝛼y by (𝛼q)y . Now when both Alice and Bob calculate the session key, they agree on ((𝛼q)x)y , which is the same for both of them.

However, this is in a subgroup of size R (assumed to be small), since it’s generated by 𝛽 = 𝛼q , as we said in (i).

## b) iii.

For Eve to find the secret session key, she needs to find the discrete logarithm of the public keys, to get x, y.

The discrete logarithm in a subgroup of size R is easy, since Eve can do trial-and-error since the possibilities are few. The discrete logarithm being hard is what the security of this protocol rests on, so it is undermined.

## c) i.

Send 𝛼x ‖ priv(𝛼x) where priv is encryption using the private key of a public-key signature scheme i.e. digital signatures e.g. RSA, ECDSA.

Then Bob gets 𝛼x and can compute pub(priv(𝛼x)) which must equal 𝛼x

Likewise, Bob can send back 𝛼y ‖ priv(𝛼y) so that Alice can check pub(priv(𝛼y)) equals 𝛼y

## c) ii.

If we have matches in both cases, it could only have come from Alice/Bob (resp.), since it required their private keys to create. This guarantees integrity/agency of the messages, and prevents someone else - Eve/MITM - from replacing/tampering with messages.

# 3.

## a)

Not assessed anymore?

## b)

113 + 7 = 12 (mod 13)

82 = 12 (mod 13)

So yes, it is within the group.

## c) i.

{A, C}, {A, D}, {A, E}, {B, E}, {C, D}

## c) ii.

{B, D, E}, {B, C, E}, {B, C, D}, {A, C, D}, {A, B, E}

## c) iii.

Compute 5 subshares:

s = s1 𐌈 s2 𐌈 s3 𐌈 s4 𐌈 s5

All l bits, just as with s. Create s1, s2, s3, s4 by random initialisation, calculate s5 = s 𐌈 s1 𐌈 s2 𐌈 s3 𐌈 s4. Thus the equation above will hold.

sA = s4 ‖ s5

sB = s1 ‖ s2 ‖ s3 ‖ s5

sC = s2 ‖ s3 ‖ s4

sD = s1 ‖ s3 ‖ s4

sE = s1 ‖ s2 ‖ s5

## c) iv.

Together A and E learn:

sA = s4 ‖ s5

sE = s1 ‖ s2 ‖ s5

That is, s1, s2, s4, s5. This is not enough to find s, as it’s missing s3.

They do have the length of each si, which matches the length of s.

# 4.

I don’t think this is assessed anymore??? Gonna give it a go anyway

## a) i.

public key: 4b\*2b = 8b2  if b=128, 131072

private key: 4b\*2b = 8b2 if b=128, 131072

signature: 2b + 2b \* 2b = 2b + 4b2 if b=128, 65792

## a) ii.

2b + 4 b2 > 8 \* 1024

2b2 + b - 4096 > 0

b > 45.006 (says wolfram alpha, ignoring -ve)

next smallest power of 2 is 64

## b) i.

Bob computes: H(r ‖ m) = h1h2 ... h2b (r is in the signature, m is the message)

Bob can then find x1[h1], x2[h2], ... x2b[h2b] from the signature

So he can compute H(x1[h1]), H(x2[h2]), ... H(x2b[h2b]) the hash of above

Which can then be compared to y1[h1], y2[h2], ... y2b[h2b] from the public key

If they match we have correctness.

## b) ii.

The public key was computed from the secret key as yi[j] = H(xi[j])

So the public key is H(x1[0]), H(x1[1]), H(x2[0]), H(x2[1]) ... H(x2b[0]), H(x2b[1])

If the message sent matches the message received, we would have the same h1h2 ... h2b

Thus H(x1[h1]), H(x2[h2]), ... H(x2b[h2b]) would match exactly

And if the message is tampered with, H(r ‖ m’) = h’1h’2 ... h’2b

Then we have distinct m’ (by property of hash function) so the signature would not match the public key

## c) i.

As the signature contains exactly half the private key. Reusing the private key would allow an attacker to have (upto) the rest of the private key. This would allow them to tamper with the message, then be able to provide a valid signature for it, using the part of the secret from the previous use.

## c) ii.

The randomness of the secret key being informationally perfect (actually uniform).

The hash function being computationally secure.

The secret key is kept secret.

## c) iii.

The hash function would be flawed. A collision is very likely, but more so, and computing the preimage is computationally feasible (by brute force), due to the small number of possibilities.

If we can find the preimage, we can get the secret key from the public key (it’s just the hash).

## c) iv.