1

1.1

key: JDQLWBSNZM
w1: MONISTICAL
w2: APHRODITES

1.2

The first step was to load all the words from the word list into a tree, where each depth of the tree corresponds with an ith letter of the word. The branches that come off each node correspond to the next letter of the word.i.e.

- aa…
- ab…
- ba…

Would generate a tree that looks like:

$$() \to (a \to (a, b), b \to (a))$$

Since the words were encrypted with the same key, that means if we were to generate a possible key, that key would need to decrypt both ciphertexts such that when the tree is navigated we navigate to nodes that exist. If the key results in a path in the tree that does not exist, then we can disregard that answer as a possible key and continue with the possible next key. Once you find a key that is the same length as the cipher text, we know that we found the right key.

2

2.1

Ciphertext: 6cea122f3b42975bdbbeb7f2c6efaf9fd5a54fdd62**3c**27**6f**55358f4fbcb7a9492d0451b7019c69faef5fd23103ff
T=2nd block
U=6th byte from the 2nd block (38th bit overall)
V=0x33
W=0x3c
X=8th byte from the 2nd block (40th bit overall)
Y=0x6c
Z=0x6f
To change the given cipher text, we need to first find the block we want to change and go to the previous block, this only works for blocks after the first one, after that, we need to find the value that comes out of the Encryption function, and we can do that if we follow this formula:

$$\text{After Encryption} \oplus \text{Previous Block Original Ciphertext} = \text{PlainText} \iff \text{After Encryption} = \text{Previous Block Ori}$$

After we calculate the value that comes out of the encryption function and before we xor with the previous block, we can now calculate the value that we need to change the previous block in the cipher text to:

$$\text{After Encryption} \oplus \text{Previous Block Altered Ciphertext} = \text{Altered PlainText} \iff \text{Previous Block Altered Ciphert}$$

2.2

The plaintext will change in the sections that we want to change; ”7:00” to ”8:30”; and ”t Guildford Stat” will change to random values.

2.3

The change is similar to the one described in 2.1 but with the iv value instead of the previous block

$$\text{After Encryption}\oplus\text{Original IV value} = \text{PlainText} \iff \text{After Encryption} = \text{Original IV value}\oplus\text{PlainText}$$

After we calculate the value that comes out of the encryption function and before we xor with IV value, we can now calculate the value that we need to change the IV value to:

$$\text{After Encryption}\oplus\text{New IV value} = \text{Altered PlainText} \iff \text{New IV value} = \text{After Encryption}\oplus\text{Altered PlainTex}$$

2.4

You cannot change the location word ”station” because the word is spread between 2 blocks, which means that to change the second part of the word ”ion”, you need to change the previos block but by changing the previous block the rest of the word ”stat” would have become garbled.

3

3.1

The computational hard problem is factorization

3.2

I used factorization to obtain the private key. After obtaining the private key, I can decrypt the cipher text and obtain ”handlebars”

3.3

I used the general number sieve[1] to factorize the public modulus and obtained:

$$p = 112546167358047505471958486197519319605436748416824057782825895564365669780011$$

and

$$q = 65802972772386034028625679514602920156340140357656235951559577501150333990623$$

with p and q I calculated

$$d = 1545653943570564246212141988589994139279645559486726912293297140150091 5$$

$$98977726717239879077953798120855868459360771804433616650588668281034152580212290153$$

with d you can decrypt the ciphertext I used the OpenSSL crypto library with the p, q, d, m, e to decrypt the cipher text

3.4

While factorizing the numbers takes more time, than a dictionary attack, it allows me to decrypt any message that was encrypted with this public key. It also allows me to decrypt messages that have different padding, including padding methods that use random values.

3.5

Yes, since I know the private key I can just decrypt the message.

4

4.1

$$P||R = E(K, C)$$

then you can remove the R part and the P can be obtained

4.2

The q pairs could look like:

$$(1, k), (2, k), (3, k) \cdots (q, k)$$

where k is a constant value for simplicity sets say $k = 0$

E_b is the list of encrypted values returned by the oracle

These q pairs work because when the oracle selects b=0:

There will be no collisions:

$$\forall i, k : i \neq j \wedge P_{0i} \neq P_{0j} \implies E_{0i} \neq E_{0j}$$

therefore if you don't find any colissions you can assume that the the oracle selected b=0

if the oracle selects b=1 and if q is big enough, there will be colissions:

$$\exists i, k : i \neq j \wedge P_{1i} = P_{1j} \wedge R_i = R_j \implies E_{1i} = E_{1j}$$

where R is the list of random values generated for each pair

4.3

Our random value is $R = u - bitlongdigit$ which means that it has 2^u possible values. And since we know that if we throw q balls into p holes, a collision is bound to happen at the probability of $\frac{q^2}{2p}$, that guessing a 2^u random value by doing q guesses is:

$$\frac{q^2}{2(2^u)}$$

We can calculate:

$$\frac{q^2}{2(2^u)} > \frac{1}{2} \iff q > 2^{\frac{u}{2}}$$

4.4

The size of TripleDES is 64 bit long which makes $u = 64/2 = 32$ making the q

$$q > 2^{\frac{32}{2}} \iff q > 65536$$

4.5

The size of AES is 128 bit long which makes $u = 128/2 = 64$ making the q

$$q > t2^{\frac{64}{2}} \iff q > 4294967296$$

4.6

Since in both 4.4 and 4.5 the value of q is not large enough, the scheme is not CPA secure

5

5.1

The hash function is collision resistant for $n = 1$, since if the block size is one of, the hash function is the encryption. Therefore: if the message is only one block long:

$$H = E$$

$$m \neq m'$$

$$H(m) = E(K, IV \oplus m) = C_1$$

$$H(m') = E(K, IV \oplus m') = C_2$$

And if the hashing function was not collision resistant, that would imply

$$C_1 = C_2 \implies D(C_1) = D(C_2) \implies m = m'$$

and since $m \neq m'$ the hash function is collision resistant, for messages with 1 block.

For if the block size is bigger than one we can say

$$H(m) = E(m)_{\text{Last Block}}$$

$$E(m) = E(K, m)$$

$$\exists a, b, c, d : m = a||b \wedge m' = c||d$$

where a,b,c,d are the size of one block

$$H(m) = E(b \oplus E(a \oplus IV)) = C_1$$

$$H(m') = E(d \oplus E(c \oplus IV)) = C_2$$

since it's possible to have:

$$b \oplus E(a \oplus IV) = d \oplus E(c \oplus IV) \implies$$

$$\implies C_1 = C_2$$

with:

$$a \neq b \neq c \neq d$$

therefore

$$H(m) = H(m') \wedge m \neq m'$$

therefore, the hash function is not collision resistant. Since this can be expanded with more than 2 blocks, the hash function is not collision resistant for any message bigger than 1 block.

5.2

When the message has the size of a block, the authenticated encryption system scheme has both data confidentiality and integrity because the hash function is only collision resistant with messages of block size 1. As a result, it is impossible to change the ciphertext in away that when the MAC is generated on the receiver side, the mac will not be the same. And since the mac key is not public, the attacker cannot generate a new mac to authenticate the fake message.

When the message has a bigger size than one block, the scheme still has data confidentiality because the message can still not be decrypted without knowing the key. But it has no longer data integrity because the attacker can change the message in such a way that it would generate a hash collision; therefore the receiver could not prove that the information that was received was not sent that way by the sender; therefore the encryption system does not have data integrity.

6

Senario 1

6.1.1

Bob can check if the equation holds then Bob knows that Alice signed the Contract

$$h = H(g^s \times y^h \text{ mod } p||C)$$

where y is Alice's pub key.

6.1.2

If the Alice used the the same r then this equation would only have 2 variables to solve, a and r which makes this equation possible to solve.

$$\begin{cases} s = r - h \times a \text{ mod } q \\ s' = r - h' \times a \text{ mod } q \end{cases} \iff \begin{cases} r = s + h \times a \\ a = \frac{s'-s}{h-h'} \wedge h \neq h' \end{cases}$$

therefor Alice private key a is:

$$a = \frac{s' - s}{h - h'}$$

Senario 2

6.2.1

To sign a contract C Alice first chooses 2 random values r and c_2 then z is calculated $z = g^r \times y_b^{c_2}$. After we have z we can calculate the intermediary value c, $c = H(y_a, y_b, C, z)$. After having c we calculate c_1, $c_1 = c - c_2$. c_1 is then used to calculate $s = r - c1 \times amodq$. The signature is (c_1, c_2, s)

6.2.2

No because Alice only needs Bob's public key which is publicly avaiable

6.2.3

The signature is verified if the equation holds

$$c_1 + c_2 = H(y_a, y_b, C, g^s \times y_a^{c1} \times y_b^{c2} modp)$$

6.2.4

No, because the signature is generated from multiple public keys and Alice's private key; therefore Chris will not be able to tell who signed the contract

Senario 3

6.3.1

The encryption works because the numbers that were chosen by Alice and Bob make this equation work

$$(m^{r_a})^{r_b} = m(\text{mod } p)$$

which means that

$$(((m^{r_{a1}})^{r_{b1}})^{r_{a2}})^{r_{b2}} = m(\text{mod } p)$$

in this case, r_{a1} from Alice cancels r_{a2} from Alice, and r_{b1} from Bob cancels r_{b2} from Bob.

6.3.2

To send an encrypted message using this system between 2 people, i.e. Alice and Bob:

1. Bob and Alice choose a prime p
2. The sender, let's say Alice, selects m and two random values r_{a1} and r_{a2} such that $(m^{r_{a1}})^{r_{a2}} = m(\text{mod } p)$
3. Alice then calculates $t1 = m^{r_{a1}}(\text{mod } p)$, Alice sends $t1$ to bob.
4. Bob selects two random values r_{b1} and r_{b2} such that $(m^{r_{b1}})^{r_{b2}} = m(\text{mod } p)$
5. Bob then calculates $t2 = t1^{r_{b1}}(\text{mod } p)$, Bob sends $t2$ to Alice
6. Alice then calculates $t3 = t2^{r_{a2}}(\text{mod } p)$, this undoes step 3, then Alice sends $t3$ to bob
7. Bob then calculates $m = t3^{r_{b2}}(\text{mod } p)$, this undoes step 5

6.3.3

Information is exchanged 4 times with this crypto system, they choose the primes and then 3 exchanges happen during the encryption process.

While for ElGamal you need to exchange information only twice, once to exchange public keys and the second to exchange the encrypted message

6.3.4

If the discrete logarithm problem is easy to solve, then Elgamal is also easy to solve. While for this case, the being able to solve the discrete logarithm problem does not help an attacker with breaking the algorithm; because the attacker only knows the result of the exponentiation and does not know the value of the base. This is not the case with Elgamal, where the base is publicly known.

The Diffie-Hellman problem also does not apply, since that problem relies on. If we know g^x and g^y being able to figure out g^{xy} but in this case the problem is slightly different. In this case the base, m is not public therefore being able to solve the Diffie-Hellman problem, does not help with this encryption problem.

7

7.1

$$v1 = (137, 312), v2 = (215, -187)$$

$$u1 = (1975, 438), u2 = (7548, 1627)$$

$$B = \begin{pmatrix} 137 & 312 \\ 215 & -187 \end{pmatrix}$$

$$U = \begin{pmatrix} 1975 & 438 \\ 7548 & 1627 \end{pmatrix}$$

A:

$$det(L) = |det(B)| = |-92699| = 92699$$

$$H(B) = (\frac{det(L)}{\|v1\| \times \|v2\|})^{\frac{1}{n}} = (\frac{92699}{\sqrt{v1_1^2 + v1_2^2} \times \sqrt{v2_1^2 + v2_2^2}})^{\frac{1}{2}} =$$

$$= \frac{\sqrt{92699}}{9427678922^{\frac{1}{4}}} \approx 0.977094 \approx 0.98$$

The Hadamard ration for the private bias is 0.98

$$H(U) = (\frac{det(L)}{\|u1\| \times \|u2\|})^{\frac{1}{n}} = (\frac{92699}{\sqrt{u1_1^2 + u1_2^2} \times \sqrt{u2_1^2 + u2_2^2}})^{\frac{1}{2}} =$$

$$= \frac{\sqrt{92699}}{243990681350077^{\frac{1}{4}}} \approx 0.0770361 \approx 0.077$$

The Hadamard ration for the public bias is 0.077
B:

$$w = (30548, 6642)$$

$$\begin{cases} 30548 = 137t_1 + 215t_2 \\ 6642 = 312t_1 + -187t_2 \end{cases} = \begin{cases} t_1 = \frac{7140506}{92699} \\ t_2 = \frac{8621022}{92699} \end{cases} = \begin{cases} t_1 \approx 77.03 \\ t_2 \approx 93 \end{cases} = \begin{cases} t_1 \approx 77 \\ t_2 \approx 93 \end{cases}$$

$$v' = 77(137, 312) + 93(215, -187) = (30544, 6633)$$

$$r = w - v' = (4, 9)$$

$$w = v' \times m + r \iff m = (30544, 6633) \times \begin{pmatrix} 1975 & 438 \\ 7548 & 1627 \end{pmatrix}^{-1} \iff m = (4, 3)$$

The plaintext is $(4, 3)$ and the $r = (4, 9)$
C:

$$\begin{cases} 30548 = 1975t_1 + 7548t_2 \\ 6642 = 438t_1 + 1627t_2 \end{cases} = \begin{cases} t_1 = \frac{432220}{92699} \\ t_2 = \frac{262074}{92699} \end{cases} = \begin{cases} t_1 \approx 4.66 \\ t_2 \approx 2.83 \end{cases} = \begin{cases} t_1 \approx 5 \\ t_2 \approx 3 \end{cases}$$

$$v' = 5(1975, 438) + 3(7548, 1627) = (32519, 7071)$$

$$w = v' \times m' + r \iff m' = (32519, 7071) \times \begin{pmatrix} 1975 & 438 \\ 7548 & 1627 \end{pmatrix}^{-1} \iff m' = (5, 3)$$

Using u_1 and u_2 we do not dectypt correctly $m \neq m'$

7.2

No, he should not. If r is not changed, then we could submit to the oracle $(1, 0)$ and $(2, 0)$ and if the oracle gives us 2 cipher texts that are the same then we know that $b = 1$ and if they are different, then we know its $b = 0$ therefore not changing the r is not secure.

References

[1] T. C.-N. D. Team, *CADO-NFS, an implementation of the number field sieve algorithm*, Release 2.3.0, 2017. [Online]. Available: `http://cado-nfs.inria.fr/`.