Question 1.

(a) part a

We know that $p = 31$, $q = 37$, $n = pq = 31 * 37 = 1147$, $e = 11$.

(i) Encryption

My UCO is 408367. We cannot use the whole UCO as a message, because it is of a higher value than n. Therefore, we divide UCO into two parts: $m_1 = 408$, $m_2 = 367$.

Then we encrypt the messages in the following way:
$c_1 = m_1^e \bmod n = 408^{11} \bmod 1147 = 149$
$c_2 = m_2^e \bmod n = 367^{11} \bmod 1147 = 564$

Therefore, the encrypted UCO is 149564.

(ii) Decryption

To decrypt, we need d such that:
$e * d = 1 \bmod [(p-1)(q-1)]$
$11 * d = 1 \bmod 1080$
Therefore, $d = 491$.

Then, we can decrypt the messages in the following way:
$m_1 = c_1^d \bmod n = 149^{491} \bmod 1147 = 408$
$m_2 = c_2^d \bmod n = 564^{491} \bmod 1147 = 367$

(b) part b

The last two digits of my UCO are 67, therefore in binary: 1000011.

We encrypt the message $w = (1, 0, 0, 0, 0, 1, 1)$ by computing $X'w^T$, so the encryption is:
$X'w^T = (393, 396, 140, 152, 435, 486, 323)(1, 0, 0, 0, 0, 1, 1) = 1202$

Then, let's decrypt. It is known that $u = 131$ and $m = 521$.

(i) First, we compute $131^{-1} \bmod 521 = 175$.

(b) part b

The last two digits of my UCO are 67, therefore in binary: 1000011.

We encrypt the message $w = (1, 0, 0, 0, 0, 1, 1)$ by computing $X'w^T$, so the encryption is:
$X'w^T = (393, 396, 140, 152, 435, 486, 323)(1, 0, 0, 0, 0, 1, 1) = 1202$

Then, let's decrypt. It is known that $u = 131$ and $m = 521$.

(i) First, we compute 131^{-1} mod $521 = 175$.

(ii) Then, we compute: $175 * c$ mod $521 = 175 * 1202$ mod $521 = 387$.

(iii) To be able to decrypt, we need X that we are able to compute from X':
$X' = u * (x_1, x_2, x_3, x_4, x_5, x_6, x_7)$ mod m
$X' = 131 * (x_1, x_2, x_3, x_4, x_5, x_6, x_7)$ mod 521

Therefore, X is equal to:
$X = (3, 7, 13, 29, 59, 127, 257)$

(iv) Finally, we are able to decrypt in the following way:
$387 > 257$, therefore x_7: 1 and our new value is 387 - 257 = 130
$130 > 127$, therefore x_6: 1 and our new value is 130 - 127 = 3
$3 < 59$, therefore x_5: 0
$3 < 29$, therefore x_4: 0
$3 < 13$, therefore x_3: 0
$3 < 7$, therefore x_2: 0
$3 = 3$, therefore x_1: 1

Therefore, the message after the decryption is: 1000011.

Question 2.

No, these moduli are not safe. We suppose that the device is not perfect and generates some primes more often then the others. Moduli are then product of two primes, but there are some common primes, therefore the moduli can have the common prime as their gcd. We can easily for every pair (or generally tuple) of generated moduli compute gcd (using euclidean algorithm) and therefore factorize the moduli without bruteforce. We can check:

gcd(65201327,134635439) = 8219, therefore we can easily compute $65201327/8219 = 7933$ which gives us $65201327 = 8219 \cdot 7933$ and $134635439/8219 = 16381$ which gives us $134635439 = 8219 \cdot 16381$.

gcd(122176133,122237737) = 15401, therefore we can easily compute $122176133/15401 = 7933$ which gives us $122176133 = 15401 \cdot 7933$ and $122237737/15401 = 7937$ which gives us $122237737 = 15401 \cdot 7937$.

gcd(122237737,99633161) = 7937, therefore we can easily compute $122237737/7937 = 15401$ which gives us $122237737 = 15401 \cdot 7937$ (we already have this in the second step) and $99633161/7937 = 12553$ which gives us $99633161 = 7937 \cdot 12553$.

There was no moduli with trivial common divisors with other moduli therefore there is no secure moduli (we were able to easily factorize all of them).

Question 3.

Solution: $633917 = 593 \cdot 1069$.

Since $n = pq$ and $\varphi(n) = (p-1)(q-1)$, we need to solve the following nonlinear system of two equations and two variables:

$$633917 = pq$$
$$633256 = (p-1)(q-1)$$

From the second equation, we can express p as $p = pq - q + 1 - 633256 = 633917 - q + 1 - 633256 = 662 - q$. Inserting this in the first equation we get

$$(662 - q) \cdot q = +633917$$
$$662q - q^2 - 633917 = 0$$
$$q^2 - 1662q + 633917 = 0$$

By solving this quadratic equation we obtain that $q_1 = 593$ and $q_2 = 1069$. This corresponds to the fact that p and q are interchangeable, and the factorisation is $633917 = 593 \cdot 1069$.

Question 4.

We need to show that for all c that g_c is strongly one-way function, that is (using definition from cr1905_2_2.pdf page 21):

- g_c can be computed in polynomial time
- there are $d_c, \epsilon_c > 0$ such that $|x|_c^{\epsilon} \leq |g_c(x)| \leq |x|^{d_c}$
- for every randomized polynomial time algorithm A, and any constant $d > 0$, there exists an m_d such that for $|x| = m > m_d : Pr(A(g_c(x)) \in g_c^{-1}(g_c(x))) < \frac{1}{m^d}$

Furthermore we know that f is strongly one-way function, so we know following:

- f can be computed in polynomial time
  - hence there is polynomial time algorithm F which computes f
- there are $d, \epsilon > 0$ such that $|x|^{\epsilon} \leq |f(x)| \leq |x|^{d}$
- for every randomized polynomial time algorithm A, and any constant $d > 0$, there exists an m_d such that for $|x| = m > m_d : Pr(A(f(x)) \in f^{-1}(f(x))) < \frac{1}{m^d}$

Let $c \in \{0,1\}^n$ be arbitrary:

We will prove the first bullet, that is, we will show that g_c can be computed in polynomial time. Consider following algorithm G_c:

- input x is binary word of length $2n$
- split x into half, call these x_1, x_2
- compute $A(x_2)$, call the result y
- concatenate c and y, call the result z
- return z

We can see that this algorithm G_c computes g_c and that it is polynomial time, because A is polynomial time and remaining operations are linear in size of input. Hence g_c can be computed in polynomial time.

We will prove the second bullet, that is, we will find $d_c, \epsilon_c > 0$ such that $|x|_c^\epsilon \leq |g_c(x)| \leq |x|^{d_c}$. Notice that this is trivial by choice $d_c = \epsilon_c = 1$, because size of input and output of g_c is $2n$, that is, $|x| = 2n$ and $|g_c(x)| = 2n$, hence $2n = |x|^1 \leq 2n = |g_c(x)| \leq |x|^{d_c} = 2n$.

Finally we will prove the third bullet, that is, we will show that for every randomized polynomial time algorithm A, and any constant $d > 0$, there exists an m_d such that for $|x| = m > m_d$: $Pr(A(g_c(x)) \in g_c^{-1}(g_c(x))) < \frac{1}{m^d}$:

We will prove this by reduction:

Consider there is algorithm A such that it would not hold, that is, there is constant $d > 0$ and for all m_d there exists $|x_m| = m > m_d$ such that $Pr(A(g_c(x_m)) \in g_c^{-1}(g_c(x_m))) > \frac{1}{m^d}$.

Then consider following algorithm F_A:

- input x is binary word of length n
- concatenate c and x, call the result y
- compute $A(y)$, call the result z
- split z into half, call these z_1, z_2
- return z_2

Let x_1, x_2 be such that $x_1||x_2 = x$, then notice that $g_c(x) = c||f(x_1)$. Hence for all $m > m_d$ there exists $x_{m,1}, x_{m,2}$ such that $x_{m,1}||x_{m,2} = x_m$ and therefore $Pr(A(g_c(x_m)) \in g_c^{-1}(g_c(x_m))) = Pr(F_A(f(x_{m,1})) \in f^{-1}(f(x_{m,1}))) > \frac{1}{m^d}$, which contradicts our assumption that f is strongly one-way function. Hence there is no such algorithm A.

That is, we have proven all three bullets, hence g_c is strongly one-way function.

Question 5.

(a)

$$G' = SGP = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

(b) $e_K(w, e) = wG' + e$
$(1010) * G' = 1010100$
$1010100 + 0000100 = 1010000$ which is our encoded word.

(c) $c_1 = cP^{-1}$. Since P is orthogonal, it's the same as $c_1 = cP^T = (1100110) * P^T = 1110010$

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

$c_1 H^T = (010)\ldots$ corresponds to sixth column of H, so $c = 1110000 \implies w_1 = 1110$
Since $w_1 = wS$, we need to find S^{-1} and compute $w_1 * S^{-1}$ to get w.

$$S^{-1} = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix}$$

$(1110) * S^{-1} = 0110$ which is the decoded cryptotext.

Question 6.

This is clearly some variation of the three-pass protocol, so after the second step, they should continue as follows:

3. Alice computes modular multiplicative inverse of e_A, let's call it d_A such that $d_A * e_A \equiv 1 \pmod{2^n - 1}$. Then she computes $C, C = B^{d_A}$ in $GF(2^n)$ and sends C to Bob. ($C = m^{e_B}$, because Alice now unlocked her "lock" on the message)

4. Bob computes modular multiplicative inverse of e_B, let's call it d_B such that $d_B * e_B \equiv 1 \pmod{2^n - 1}$. Then he can get m by computing C^{d_B} in $GF(2^n) = m$.

If m was, for example, some encryption key, now the two of them can begin encrypted communication using m.

Proof:
First, consider the case where $m > 0$. All nonzero elements of $GF(2^n)$ form a multiplicative group of order $2^n - 1$. From Lagrange's theorem we know that the order of group element divides order of finite group. In our case this means that for some m in $GF(2^n)$ if x is the smallest nonzero integer

such that $m^x = 1$ in $GF(2^n)$ then x must divide $2^n - 1$. With the previous argument in mind, we also know that $m^{2^n-1} = 1$ in $GF(2^n)$. It follows that $m^{k(2^n-1)} = 1$ in $GF(2^n)$ as well, and that $m^{k(2^n-1)+1} = m$ in $GF(2^n)$.

This implies that to get m from m^e in $GF(2^n)$ we need to find some integer d such that $m^{ed} = m^{k(2^n-1)+1}$ in $GF(2^n)$, so we want to satisfy the equation $de = k(2^n - 1) + 1$, that is $de \equiv 1 \pmod{2^n - 1}$, in other words we need to find the modular multiplicative inverse of e with respect to modulus $2^n - 1$. This is exactly what we are doing in steps 3. and 4 when computing d_A and d_B. We can also be sure that in our case the modular multiplicative inverse of eA or eB with respect to modulus $2^n - 1$ always exists, because we have previously enforced that $gcd(e_A, 2^n - 1) = gcd(e_B, 2^n - 1) = 1$.

To show how it works in practice after the second step:

- In the third step Alice knows e_A and $B = (m^{e_A})^{e_B} = (m^{e_B})^{e_A}$ in $GF(2^n)$ She computes d_A as described above, then computes $C = B^{d_A} = (m^{e_B})^{e_A d_A} = (m^{e_B})^{k(2^n-1)+1} = m^{e_B}$ in $GF(2^n)$.
- In fourth step Bob knows e_B and $C = m^{e_B}$, he computes d_B as described above, then $m = C^{d_B} = m^{e_B d_B} = m^{k(2^n-1)+1} = m$ in GF(2^n). He successfully received m.

Lastly, the case where $m = 0$ wasn't previously considered, but $0^x = 0$ in $GF(2^n)$ for any $x > 0$, so in this case it works as well.