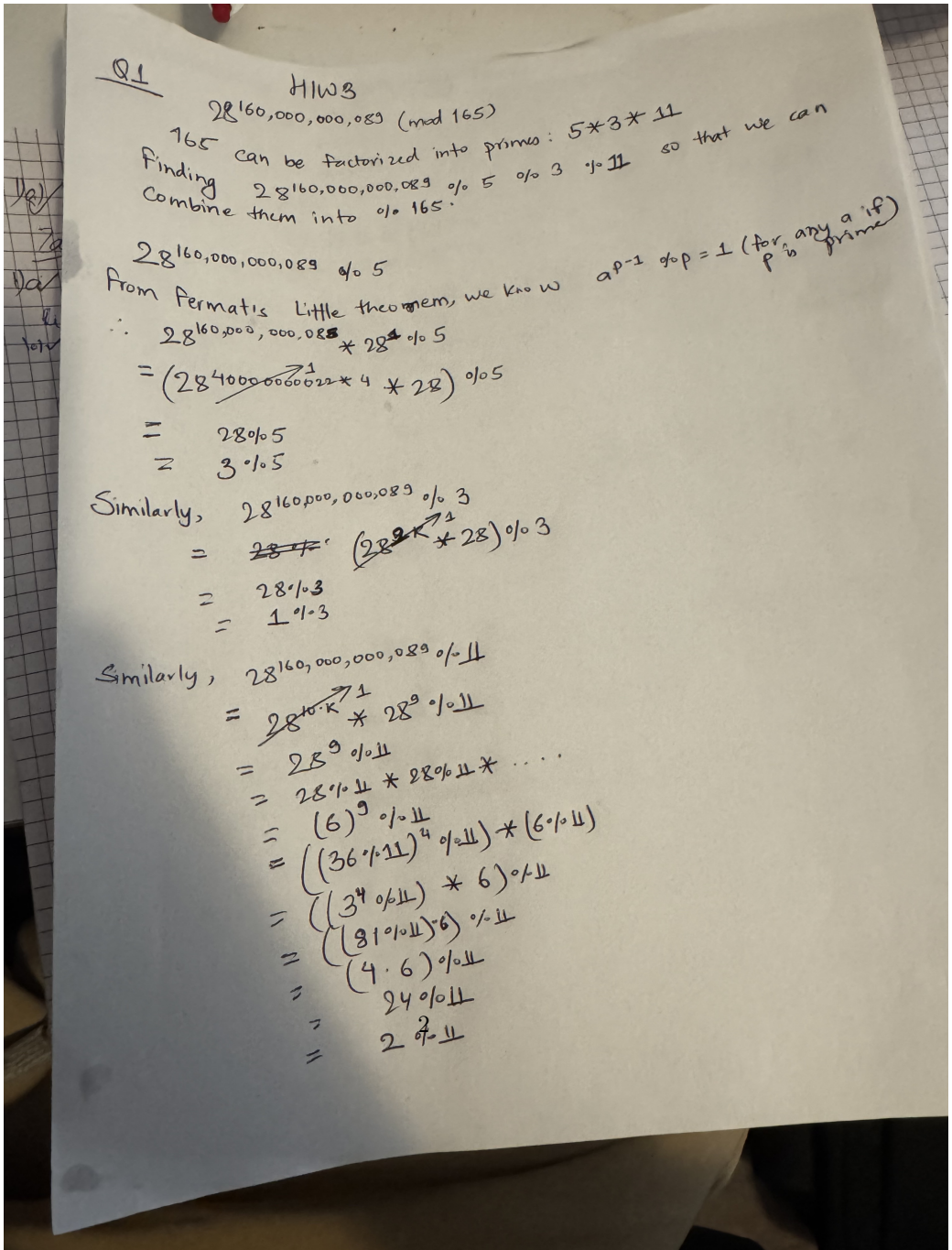
Homework 3 (CSCI-B 504)

Kushal Pokharel

October 2025

1 **[10 points] Without using a computer or any electronic devices, i.e., manually, calculate $28^{160,000,000,089}(mod 165)$. You need to list all the steps of your computation, including, for examples, co-primes used in CRT, multiplicative inverses, and all ri, qi, ti, si when computing inverses. Hint: You may need to use Chinese Remainder Theorem and Fermat's Little Theorem.**



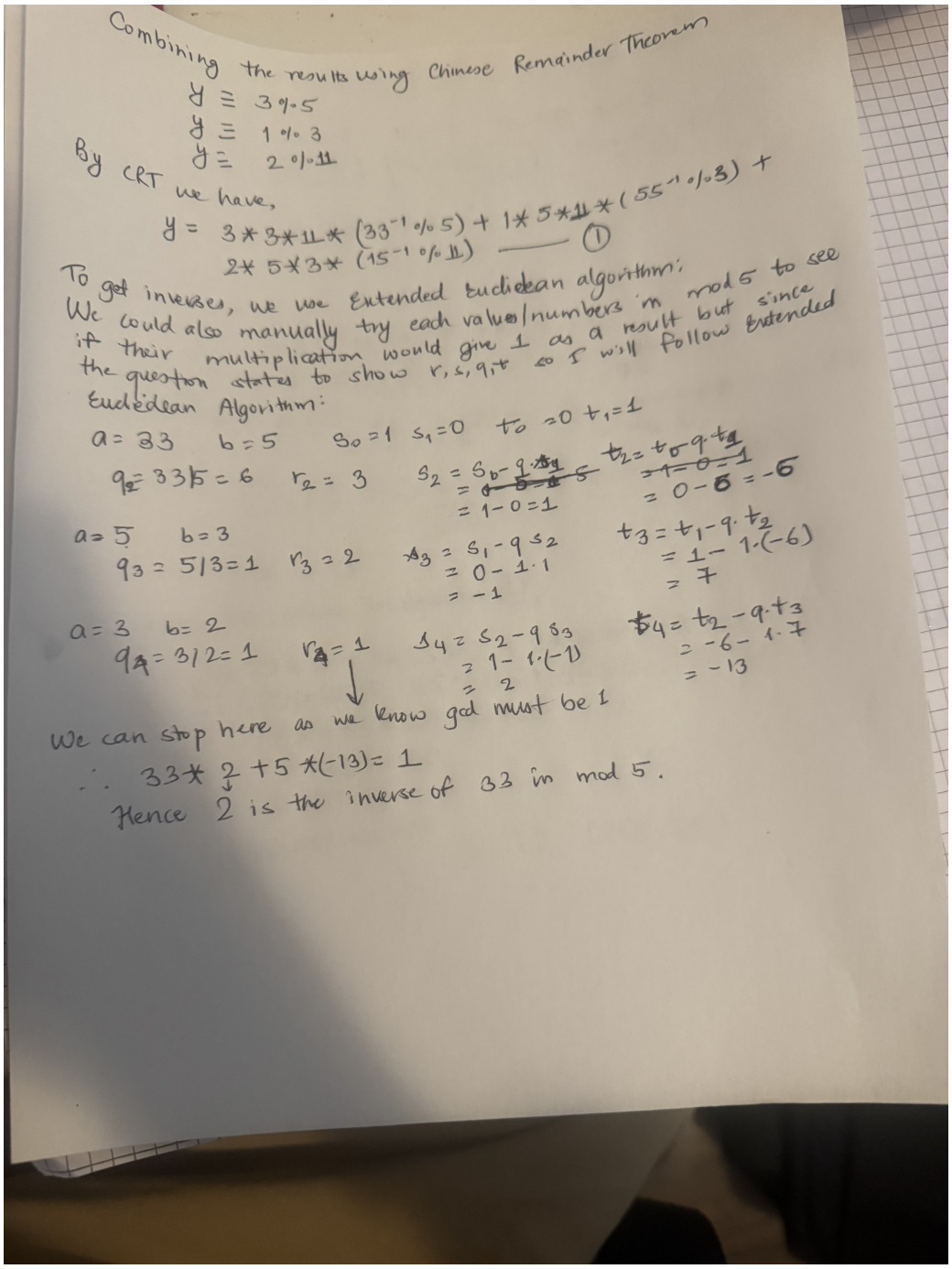


Figure 2: Page 2

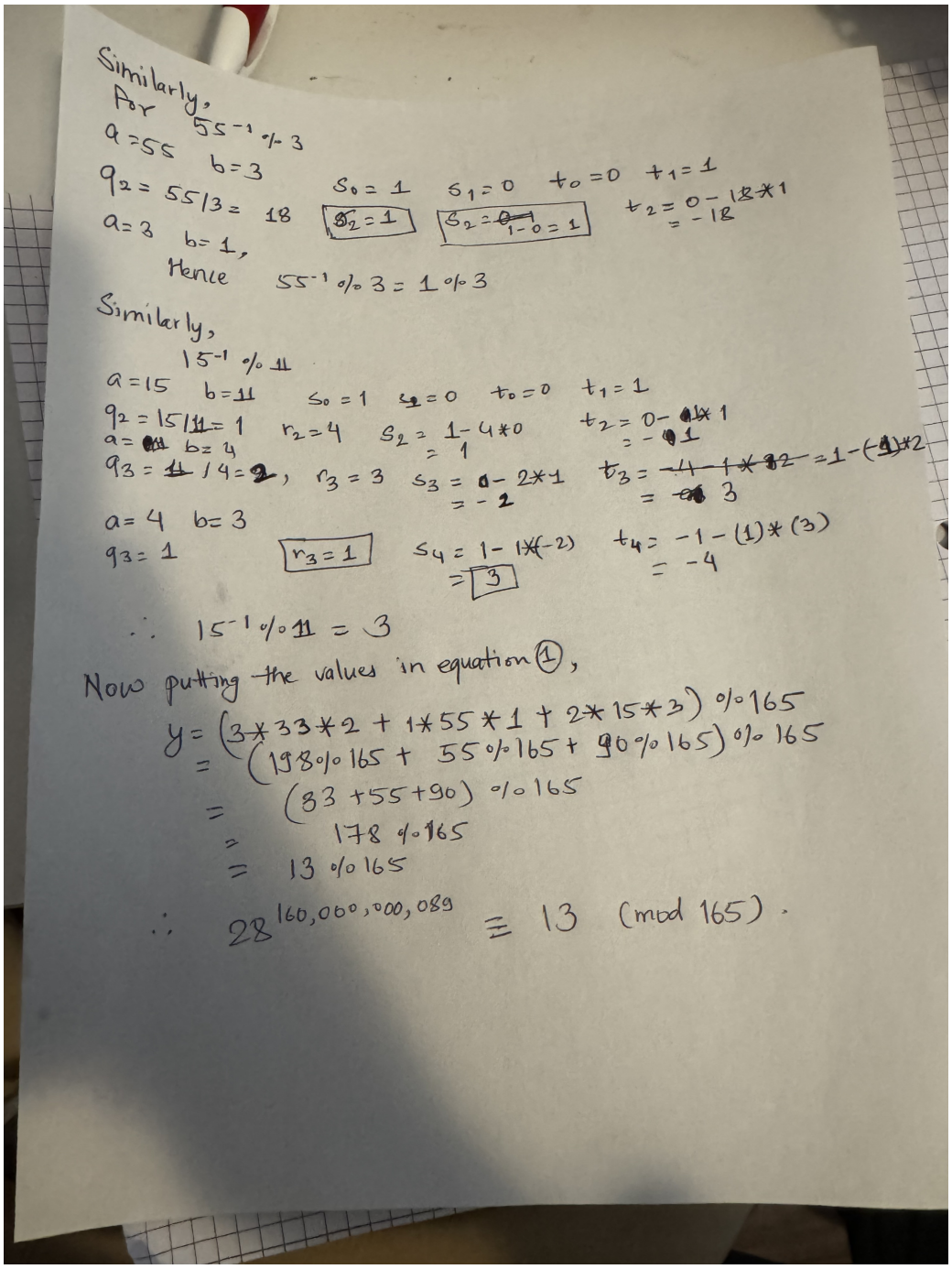


Figure 3: Page 3

2 (This is Problem 6.9, 2nd edition, or 7.9 4th edition) Decrypt the ElGamal ciphertext presented in Table 6.3 of page 270 (2nd edition) or Table 7.4 of page 305 (4th edition). You can also copy the ciphertext from cipher problem6.9 Table6.3.txt The parameters of the system are $p = 31847, = 5, a = 7899$ and $= 18074$. Each element of Z_n represents three alphabetic characters encoded as in the following examples: $DOG \rightarrow 326^2 + 1426 + 6 = 2398$
$CAT \rightarrow 226^2 + 026 + 19 = 1371$
$ZZZ \rightarrow 2526^2 + 2526 + 25 = 17575$
You will need to invert this process as the final step in your program.

2.1 Approach

Since all of the parameters were given, I only raised the given ciphertext's first component by a, found its inverse in p, and multiplied it by the second component to recover the message. Finally, the message was converted back to the characters by repeatedly dividing by 26^5, its remainder by 26^4, and so on.
I didn't find any use of alpha and beta given as the parameters.

2.2 Decrypted Plaintext

she sta nds upi nth ega rde nwh ere she has bee nwo rki nga ndl ook sin tot hed ist anc esh eha sse nse dac han gei nth ewe ath ert her eis ano the rgu sto fwi nda buc kle ofn ois ein the air and the tal lcy pre sse ssw ays het urn san dmo ves uph ill tow ard sth eho use cli mbi ngo ver alo wwa llf eel ing the fir std rop sof rai non her bar ear mss hec ros ses the log gia and qui ckl yen ter sth eho use

3 Consider ElGamal public key cryptosystem with public key (p, α, β) and private key a where $\beta = \alpha^a$. The ElGamal Signature Scheme works as follows: picks k at random from Z_{p-1}^* and computes the signature for message m as sig(m) $= (\gamma, \delta)$ where $\gamma = \alpha^k$ and $\delta = k^{-1}(m - a\gamma) mod(p - 1)$.

Prove that such a signature scheme is not allowed to have $\delta = 0$. That is: if a message were signed with a "signature" in which $\delta = 0$, then it would be easy for an adversary to compute the private key a. Show how?

As given, If $\delta = 0$, then $m - a\gamma$ (mod p) $= 0$ since k^{-1} can not be 0, and furthermore, $a = m * \gamma - 1$. Both m and γ are input to the signature verification. Hence, a (secret key for the user) is revealed.

4 Suppose Elliptic curve E is defined by $y^2 = x^3 + x + 6$ over Z_{1039}. Since Z_{1039} is small, you can program to solve all the following problems.

4.1 How many points are over E?

4.1.1 Approach

First, brute forced across all the possible x and y in modulo p(1039) and approved those x and y that fit into the equation. Second, since 1039 is a prime and $1039\%4 = 3$ we can use $a^{\frac{p+1}{4}}$ to calculate the square root of y^2 in modulo p.
There are 1008 points in E.

4.2 What is the lexically largest point over E. Here lexically larger point means to order the points by the first coordination first and then the second coordination, E.g., $(320, 100) > (319, 500)$ and $(320, 100) < (320, 101)$.

(1038, 1037) is the lexically largest point over E.

4.3 Does point (1014, 291) belong to E?

No the point doesn't belong to E. (1014,749) and (1014,290) belongs to E.
$291^2 \% 1039 = 522$
$1014^3 + 1014 + 6 = 1042591764 \% 1039 = 980$

4.4 Suppose $\alpha = (799, 790)$ is a generator and $\beta = (385, 749)$. (E, α, β) is the ElGamal public key. Given the plaintext value (575,419) and random K=100, what is the ciphertext value? Given the ciphertext value ((873,233), (234,14)), what is the plaintext value (Please note: if you re-encrypt the resulting plaintext value with K=100, you may not get back ((873,233), (234,14)), that is correct. As you recall, for whatever K is selected for encryption, the decrypted value is the same.)

4.4.1 Approach

The first question just demands implementing the ElGamal cryptosystem in ECC, which is defined in addition, so all of the powering will reduce to multiplication, and multiplication reduces to addition. After implementing the point addition, the ciphertext corresponding to the plaintext (575,419) is found to be **((873, 233), (963, 817))**.
For the second question, we need to get the secret key using alpha and beta. Since our E is small, we can brute-force by repeatedly adding alpha and checking if it equals beta for some value of secret key (x). I didn't even have to use the Double and Add algorithm. The secret key was found to be **939**. The plaintext was then found by multiplying the first cipher by the secret key and then subtracting it from the second cipher. The corresponding plaintext is **(319, 784)**. I also found what k was used for the encryption above, and found k=100 was used, even though any k could be used for decryption to work. For the final confirmation, I encrypted the calculated plaintext with k =100 and got the ciphertext that was given in the question.

4.5 Suppose E and a generator $\alpha = (818, 121)$ are public. Alice and Bob achieve a shared secret by doing Diffie-Hellman key exchange. Alice sends Bob a value (199,72), and Bob sends Alice a value (815,519). What is the secret they achieve?

4.5.1 Approach

We need at least one of Alice's or Bob's private keys, which can be done by repeatedly multiplying the generator and comparing it with their public keys. I found Alice's secret key, which is:

Alice's secret key: 516
Shared key: (855, 475)
To verify, I also calculated Bob's private key and calculated the shared key, which should match
Bob's secret key: 1001
Shared key 2: (855, 475)

4.5.2 Screenshot of the output of the program

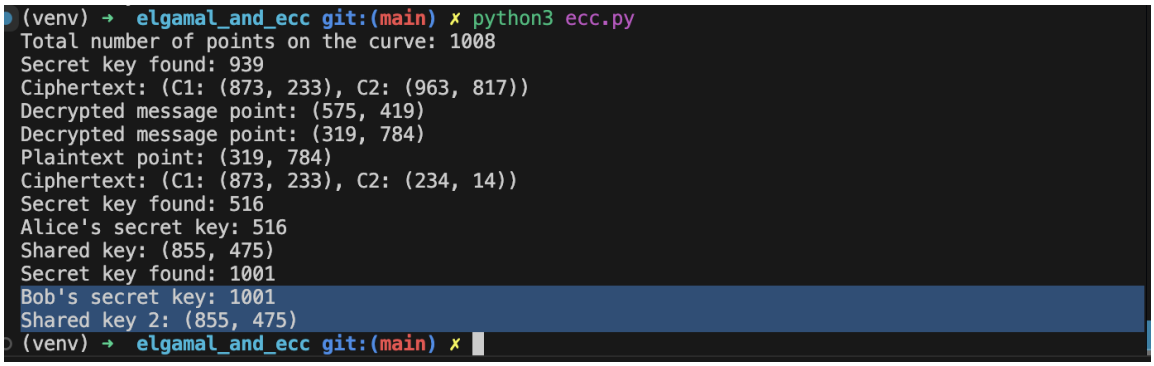


Figure 4: Output of the ECC program

5 Besides being used for message authentication and efficient signature of long message, cryptographic hash functions have many other usages such as user authentication between two users (e.g., Alice and Bob) having a shared secret key K, committing a value first and then disclosing the value later, and many security and key management protocols . One sce- nario works as follows (hash function H is public, as normally assumed):

• Alice wants to authenticate Bob, Alice selects a random r1 and sends r1 to Bob.

• Bob selects a random r2, computes z = H(r1 + K + r2), and sends (r2, z) back to Alice.

• Alice can recompute H(r1 + K + r2) and check whether it equals to z. If yes, Bob is authenticated.

Such a user authentication protocol works but is complicated, or to say, not efficient. Can you design a more efficient user authentication protocol between two users by simply using the shared secret key K and the public hash function H?

5.1 Approach

My first thought was to reduce at least one communication in this two-round communication protocol. Alice doesn't send Bob any message. Bob just picks up a random r and computes the hash as z = H(r+K) and sends (r,z) to Alice. Alice can then verify by computing the hash by combining r and K and matching with z.

However, this approach seems vulnerable to a replay attack where the adversary gets access to both of these values and can impersonate Bob at some later time. To prevent this replay attack, Alice can store the random values used by Bob and not authenticate Bob if he uses the same value twice. This would reduce

the communication time and also prevent replay attack however Alice and Bob would both need to keep track of the random values used till now using some Set data structure
We can further simplify this by introducing a concept of **nonce** where Bob's nonce starts with 1 and keeps incrementing every time Bob authenticates and sends the hash of key K + nonce (H(K+nonce)). Alice would also keep track of this one number and add this with the key K and obtain the hash and verify the hash. This prevents the replay attack and also decreases the amount of numbers that these parties have to store. Alternatively, we could also keep tiemstamp or any other values that move in one direction to prevent replay attack.

5.2 Improvement in protocol

One round of communication is reduced as Alice doesn't have to send r1 to Bob. In the given protocol, Alice had to keep track of the r1s used so that it is not vulnerable to a replay attack, which increases linearly with the number of authentications Bob has done. Comparing this to our nonce-based protocol, it only needs to store one value, which doesn't increase with communication rounds.

6 This problem is about the security services each protocol provides. Let us consider four security services: Confidentiality (C), Integrity(I), sender Authentication (A), and Non-Repudiation (NR). Notes: assume that the public key (and its private key) cannot be forged and is authentic. So, if a signature of a message can be verified via the corresponding public key, the sender will not be able to deny having sent the message. Suppose the following notations are used: k_1, k_2: keys shared between a sender S and a receiver R; $E_k(x)$: (symmetric or asymmetric) encryption of x under k; $SIG_k(x)$: signature on x under k; X_{pri}: private key of entity X; X_{pub}: public key of entity X; H: a public secure cryptographic hash function such as SHA 1; $PRNG_s$: a binary stream from a pseudo-random number generator seeded with s; $||$: simple concatenation; and M: the message. For each of the following protocols, use C, I, A, and/or NR to represent the service(s) a protocol provides. If a protocol can not provide any service, use "None".

6.1 (a) S generates a random session key s_k and sends $E_{S_{pub}}(sk)||E_{R_{pub}}(s_k)||(MPRNG_{s_k})$ to R.

Answer: C

Confidentiality (C): Because only R and S can decrypt the session key s_k and then can generate the random number from $PRNG(s_k)$, which when XORed with the third component gives them the message M and to noone else.

No Integrity: Because if the second component is altered, R has no way of knowing if it has been altered and will recover a random s_k, which in turn will recover a random message M.

No sender authentication: Anyone can send a message in such a format to R,

and R doesn't know if it came from S or not. No Non-Repudiation: Since only Spub is used for the message, S can deny that it generated these messages, as anyone might have created and sent the message in such a format.

6.2 (b) S sends $y = E_{k_1}(x||H(k_2||x))$ to R.

Answer: CIA
Confidentiality (C): Since the message is encrypted with k_1, which we assume is shared between S and R only.
Integrity(I): If anyone modified y, after the decryption with key k_1, R gets x and the hash of the message $k_2||x$, if the message was modified, it is expected that the hash of the key k_2 and x will not match the second component of the decrypted y. Sender authentication(A): If y was not modified and the S's computed hash matches the y's second component, then we know that the sender knows both k_2. k_2 included in the hash function will authenticate S. But this protocol seems vulnerable to a replay attack, in which the attacker sends the same y to impersonate someone who knows k_2 No Non-Repudiation: Since both Alice and Bob know k_1 and k_2, it doesn't ensure non-repudiation for either of those parties, as either might have generated such y.

6.3 (c) S sends y = $E_{R_{pub}}(x||SIG_{S_{pri}}(H(x)))$ to R.

Answer: C,I,A,NR
Confidentiality (C): The payload is encrypted with R_{pub} which means only R can decrypt the message. Hence, confidential.
Integrity(I): If anyone modified y, after the decryption with key R_{priv}, R gets x and the signature of $H(x)$) signed with S's private key, if the message was modified, it is expected that the hash of the message x (first component) when verified with S's signature, will not be verified. Hence, the receiver knows it was tampered Sender authentication(A): If the signature's verification outputs true, then we know that it was signed by the sender, hence we know that it was sent by the sender(who possesses S_{priv}) and no one else. Non-Repudiation(NR): Since the signature is included, the sender can not repudiate sending this message later on.

6.4 (d) S generates a new symmetric key sk and sends $y = E_{S_{pub}}(s_k)||E_{R_{pub}}(s_k)||SIG_{S_{pri}}(s_k)||E_{s_k}(x)$ to R.

Answer: None
Any third party can get s_k from the signature using the third component (S's signature over s_k) using S's public key (since it is public), and finally can use that s_k from the fourth component to get the x by decrypting it with s_k.