General Advice

Read the question carefully and:

- Follow the description in the question with respect to terminology.
- If you are uncertain about details, spell this out clearly in your answer. For example, if you are not sure what are the assumptions about the threat model in the question, state in your answer how you interpreted the question.
- Make sure you answer all subquestions.

Question: Memory leaks (all versions)

Error 1: Confusing the direction in which data is written on the stack. Many try to overwrite variables inserted in the stack *after* the variable on which there can be an overflow. This will not work, as data is inserted in the stack in the other direction.

Error 2: Forgetting to check whether overwritten variables are updated. Many use an overflow to overwrite a variable, and assume that the value assigned in the overwrite will stay without analyzing the rest of the program. If the overwritten variable is assigned a new value before the use, then the exploit cannot be successful.

Error 3: Forgetting that to overwrite a return address more things need to be overwritten. Many use an overflow to overwrite a return address. While doing this they do not consider that between the variable they are overflowing and that return address there are more variables that may make the program fail or run into undefined behaviour. Unless there was an explicit indication of how the return address would be overwritten without damage these answers were penalized.

Error 4: Forgetting negative indexes. When checking bounds for variables, many forgot to check overflow in the direction of the start of the array. Negative indexes also access memory that is not reserved.

Question: Hacking NotSoSec

Error 1: Incorrect description of non-CSRF attack. Some of you described attacks that are not Cross-site Request Forgeries (for example, brute force, XSS, etc.). The descriptions of how these attacks were carried out did not apply to the scenario described in the system. Note that if you stated your assumptions and the attack was plausible based on the scenario and the assumptions, you would receive points.

Error 2: For a CSRF, it is the cookies of the victim site that are relevant. When describing the session cookies, it is the cookies of the director on the bank's site (stored in the browser) that are relevant to the problem, and are used in the checks provided in the code snippet. It is not the cookies of the director on the weather site.

Question: Authentication mechanism (all versions)

Error 1: Storing a password as a salted hash prevents dictionary attacks. Storing passwords as a salted hash with a different salt for each password, does not entirely prevent an attacker from revealing a user's password. An adversary who gets access to the bank's database could still brute-force a user's password given the hash function and the specific salt stored alongside the password. Brute-forcing *one* user password has the same cost as brute forcing without a salt (the adversary needs to run the dictionary with the salt associated to that password). What salts prevent are massive attacks in which with one computation the adversary can break many passwords at the same time, as they now have to repeat the attack for every new password she wants to crack.

Error 2: No discussion of usability or security aspects. Some have not explicitly discussed either usability of security aspects of one of the options, or did not justify them. Please read the question carefully.

Question: Cryptographic Exchange (all versions)

Error 1: *It is possible to extract the signed message from a signature.* Many answers assumed there is an extraction or "decryption" functionality in the signature. Digital signatures consist of two algorithm: Sign (creating a signature on a message) and Verify (verify the signature is valid). Verify algorithms do not generally enable to recover the message.

Note: Formally, a signature scheme does not provide confidentiality. In practice, however, extracting messages from most signature schemes is as hard as computing first pre-image for hash functions. If you can guess/brute-force the message then you can verify your guess, but otherwise you cannot extract the message. Unless you specifically mention that you are assuming a digital signature known to not provide confidentiality, we did not accept "extraction of a message from a signature" as there is no decryption procedure in signature schemes.

Error 2: Misunderstanding of asymmetric cryptography. Some answers did not use private and public keys correctly. In a signature scheme, a signature is created using the signer's *private key* (which is private by definition), and is verified using the signer's *public key* (which is publicly known by definition). It cannot be the other way around. In an asymmetric encryption scheme, a ciphertext is created using the receiver's *public key*, and can be decrypted using

the receiver's *private key*. Similarly, it cannot be the other way around.

Error 3: Using "encode" and "decode" instead of encrypt and decrypt. As noted in the lecture, "Encode" and "decode" are different from encryption and decryption in that encoding/decoding require only knowledge of an algorithm, while encryption/decryption require knowledge of an algorithm and a key. Using encoding to describe encryption algorithms is incorrect (if messages would be encoded, the system would be insecure).

Error 4: Forgetting that a signature ensures integrity. Some students wrote that there is no integrity in the protocol "k1, Enc(pkb, k2), Stream(k1 xor k2, m), Sign(ska, m || k1)" as there is no MAC on the message. However, when Alice signs "Sign(ska, m || k1)", the signature ensure the integrity of both m and k1 besides non-repudiation.

Error 5: You can have non-repudiation for a message which does not guarantee integrity. Some student wrote that the protocol "Enc(pkb, k), Sign(ska, k), Stream(k, m)" provides non-repudiation despite having no guarantee for integrity. The adversary can modify "Stream(k, m)" and there is no integrity check to detect this. The signature ensures that the key k is coming from Alice, but Alice can argue that "Stream(k, m)" is not her message and an adversary has changed it in transit, or Bob has created a new message m' after receiving an original message m and a signature on k.

Error 6: Non-repudiation of having a conversation. Many answered that Alice cannot repudiate a message because there is a signature in the exchange.

The question asked about Alice being able to repudiate "the message", which is explicitly the specific message m sent to Bob ("*Alice sends a message m to Bob as follows:*")

The fact that Alice may not be able to deny having a conversation with Bob, as there is a signature signed with her private key in the exchange, does not mean that she cannot repudiate the message m if this message is not protected by the signature.

Error 7: Treating "Stream(k, m)" as random string generator. The problem specifies that "Stream(k, m)" is a encryption the message m with the key k. This is an encryption function which should not be confused with the internal mechanism of how stream ciphers work. In the lecture slides, we explain the internal mechanism of stream ciphers as creating a random string based on the key and the IV as "Stream(K, IV)" then xor it with the message.

There is a name similarity between the slides and the question, but the definition and parameters of functions in the question are clear.

Error 8: Length of the one-time pad key. What matters for the security of a one-time pad is that the random key is at least as long as the message. Any excess bits of the key can be discarded, or, equivalently, the message can be

padded to the length of the key. Thus, the fact that the key is longer than the message is not a problem.

Error 9: Message repetition in a one-time pad. Two ciphertexts encrypted with a one-time pad are vulnerable to frequency attacks if the key is reused. But, there is no issue in sending a message m multiple multiple times as (m xor k1) xor (m xor k2) only reveals information about k1 xor k2 for two random keys.