CSEP590 : Applied Cryptography: Homework 6

Karuna Sagar Krishna

May 30, 2023

Task 1

The S/Key identification scheme updates the secret key (SK) and verification (VK) on each interaction between the prover and verifier. In other words, the keys are one-time use only.

Assume an eavesdropper obtained $t = H^i(K)$ sent by the prover to verifier. Firstly, this t value cannot be used again since the VK gets updated by the verifier. This t cannot be used to figure out the next value that a prover would have to send for successful verification i.e. $H^{i-1}(K)$. This is because hash functions are one-way functions that provide pre-image and second pre-image resistance properties. Note that $SK = (K, i)$ is kept secret, so the eavesdropper doesn't know neither the random string K nor the index i to compute t themselves. And even if VK is public, due to pre-image resistance, we cannot predict t that will be accepted by the verifier. Hence an eavesdropper cannot impersonate the prover.

Further, it seems in case the verification fails, the prover would have to re-register with the verifier. This is because the prover would have updated the secret key to $(K, i-1)$ before receiving the result from verifier and the verifier would not change VK. So any subsequent interactions would fail verification, unless the prover re-registers with the verifier or updates the secret key by incrementing the index. If we assume the re-registration option, it essentially locks out the prover and any eavesdropper. Though it is an usability problem it improves security.

Task 2a

As hinted, RSA satisfies homomorphic property. So consider the following:

1. The verifier generate random number $r \in Z_N$ and sends $c = r^e \mod N$
2. PITM intercepts the communication, generates $a \in Z_N$ such that $b = a^{-1} \mod N$
3. PITM then sends to prover $\widetilde{c} = (r^e \mod N) \cdot (a^e \mod N) = r^e \cdot a^e \mod N = (r \cdot a)^e \mod N$. Note $VK = (N, e)$ is public.

4. The prover now computes $r' = \widetilde{c}^d \mod N = (r \cdot a)^{ed} \mod N = (r \cdot a) \mod N$ using RSA key lemma

5. PITM intercepts and instead sends to verifier $\widetilde{r'} = (r' \cdot b) \mod N = ((r \cdot a)) \cdot b \mod N = r \mod N$ since $b = a^{-1} \mod N$

6. The verifier receives $\widetilde{r'}$ which is equal to r and hence accepts the prover

Clearly, the messages sent by the verifier is different from the one received by the prover and vice versa. The PTIM modified c into $\widetilde{c} \neq c$ and r' into $\widetilde{r'} \neq r'$ but still the verifier accepts. Hence the protocol is broken as stated in the question.

Task 2b

As seen in class, RSA is hard to invert i.e. given N, e and y it is hard to find x such that $x^e = y \mod N$. Further, the protocol suggests the verifier generate a fresh random number for each communication with the prover. This avoids security issue due to plain RSA being deterministic.

That said, there is a case where we could use the homomorphic property to impersonate the prover. Consider the following: say we record all messages between the prover and verifier. And say we received a fresh challenge $c = r^e \mod N$ where $r = r_1 \cdot r_2$ and we have previously generated r_1 and r_2. In that case we have $c = (r_1^e \mod N) \cdot (r_2^e \mod N)$. Since we have previously seen $(r_1^e \mod N)$ and $(r_2^e \mod N)$ then we know r_1 and r_2 using which we can recover r as $r_1 \cdot r_2$ which would be accepted by the verifier. Ofcourse, this would only work if we have the random number generated is a product of two previously seen random numbers. With large N, the probability of this starts low but increases as we see more random numbers exchanged between the prover and verifier. If N is sufficiently large the probability of this seems quite low. So we could say the protocol is secure against eavesdropper.

Task 2c

If Π is IND-CCA secure, then by definition a PITM cannot gather any useful information by changing the encrypted challenge c sent by verifier to prover. Specifically, though the PITM adversary might have access to both encryption and decryption oracles, it cannot learn anything from the challenge ciphertext c sent by the verifier.

On receiving c, the prover would decrypt and send it in plain text to verifier. Here the PITM can intercept and modify the message. However, any changes will only result in rejection by the verifier since the value needs to match the random number r held secret by the verifier. Hence the protocol is secure against PITM.

Task 3

One of the main differences between AES and hash function is that AES is reversible while hash functions are one-way and hard to reverse. So using AES with fixed key 0^{128} to compute the next state of the entropy pool implies that we can find the previous state S_{i-1} of the entropy pool given the current state S_i.

Lets say the current state S_i was computed by the generate algorithm and this state is compromised. Then the adversary can compute the previous state S_{i-1} by using inverse AES function with fixed key 0^{128}. With S_{i-1}, we can use SHA-512 to determine the random number Z_i handed out by generate algorithm. This violates forward secrecy property. Further, if we have a chain of states that were computed using generate algorithm, then we can compute all the corresponding random numbers handed out.

Similarly, if current state S_i is compromised and the say there is a chain of states that are computed by generate algorithm then the random numbers handed out by each of them can be predicted. This is because the next state can be easily calculated using AES and fixed key 0^{128}. This violates post compromise security property.

Hence, using AES with fixed key in generate algorithm to compute the next state violates forward secrecy and post compromise security properties of RNG.

Task 4a

AES is a block cipher and can be reversed. However, it offers no integrity or authenticity checks. So during evaluation, we will be able to decrypt all the entries in the garbled table given A_x and B_y. AES does not return $\perp$ for any inputs. This makes it impossible to evaluate the garbled circuit.

If we consider a single gate, we could have Alice (generator) share the labels for output wire i.e. share both C_0 and C_1. Then the evaluator can check if the decryption of the garbled table entries matches either C_0 or C_1. This is inefficient since we have to perform AES decryption completely for all entries in the table and check if it is valid. Further, this becomes more complex and inefficient when we have multiple gates as in a circuit as we would have to share the labels for each gate output wire.

Task 4b

The new approach proposed by Alice prevents the problem since the label $C_{G(x,y)}$ is not encrypted using AES. However, the generator garbles by running AES on fixed known strings (0^{128} and 1^{128}). In this scheme, the evaluator first computes $(AES_{A_x}(0^{128})||AES_{A_x}(1^{128})) \oplus (AES_{B_y}(0^{128})||AES_{B_y}(1^{128}))$. Then it xor-s this with entries in the garbled table. If this result has 0^{128} as suffix, we consider this as valid decryption and output the first 128 bits as $C_{G(x,y)}$.

Otherwise, we output $\perp$. This provides a robust way to evaluate the garbled circuit.

Task 5a

Given A_x and B_y, we can find the index of the garbled table that we are interested i.e. $C_{G(x,y)}$. First we find the first bit of A_x and B_y and use that to find the row in the garbled table i.e. $C_{A_x[0]B_y[0]}$. Since the garbled table is ordered, we can find this entry in constant time. We can then decrypt it using A_x first and then using B_y to recover the label $C_{G(x,y)}$.

Task 5b

In class we saw that having an ordered garbled table leaks information about x and y. This was because the garbled table was ordered by (x, y). We fixed this by randomizing the order of the garbled table.

However, in this question, the garbled table is ordered by $(A_x[0], B_y[0])$ i.e. using the first bit of the input labels. The probability that $A_0[0]$ is 0 is 1/2 and similarly probability that $B_0[0]$ is 0 is 1/2. So the probability that C_{00} matching $x = 0$ and $y = 0$ is 1/4. Similarly, we can talk about other rows of the garbled table. Due to this we can view the garbled table as randomly shuffled. Hence the fixed order of $C_{00}, C_{01}, C_{10}, C_{11}$ does not violate security of garbling scheme.

Task 5c

Though A_0 and A_1 different by their first bit only, AES security is not compromised when using these as keys. AES will continue to act as a pseudo random function and produce random looking output hence maintaining the security of garbling scheme.

The first bit together with the ordering of the garbled table encodes the index of gate evaluation in the garbled table. As seen above, this index and ordering does not reveal any information about x or y. Since we have determined the index, we don't have to attempt to decrypt all rows in the garbled table making it more efficient while solving the problem we encountered in task 4a.

Task 5d

During evaluation, to determine the output label, the approach mentioned in task 4b requires use to pre-compute using four AES operations, xor-ing this with each row of the garbled table and then verifying the suffix. However, the new approach is more efficient since we find the index by simply looking at the first bit of A_x and B_y and then decrypting the corresponding row in the garbled table. We don't need to perform any operations on other rows. Hence the new approach is more efficient.

Similarly, the new approach is efficient for the generator as well since we don't need to randomly shuffle the garbled table.

In the new approach, each entry in the garbled table is 128 bit long while this is doubled by the approach described in task 4b. This implies the size of the garbled table is doubled. Hence the new approach is more efficient in terms of space and transferring this garbled table over the network to the evaluator.