ETH Zurich, Department of Computer Science Dr. Martin Hirt
SS 2017 Chen-Da Liu Zhang

Cryptographic Protocols
Solution to Exercise 2

2.1 Complexity Theory

a) Let us denote $\epsilon(n)$ the failure probability of A for an input of size n. Since the failure probability of A is negligible, we have that $\forall c \in \mathbb{N}\ \exists n_0 : \forall n \geq n_0 : \epsilon(n) \leq \frac{1}{n^c}$. For the algorithm A' that invokes A polynomially many times (say less than $n^{c'}$ times for some c'), the failure probability of A' is bounded by $n^{c'} \cdot \epsilon(n)$. Hence, we have that $\forall c \in \mathbb{N}\ \exists n_0 : \forall n \geq n_0 : \epsilon'(n) \leq n^{c'}\epsilon(n) \leq \frac{n^{c'}}{n^c} = \frac{1}{n^{c-c'}}$. This means, that $\forall d \in \mathbb{N}\ \exists n_0 : \forall n \geq n_0 : \epsilon'(n) \leq \frac{1}{n^d}$ and A' has negligible failure probability.

b) Recall that a function $f : \mathbb{N} \to \mathbb{R}^+$ is *negligible* if it decreases (to 0) faster than the inverse of every polynomial, i.e., if

$$\forall c \in \mathbb{N} : \exists n_0 : \forall n \geq n_0 : f(n) \leq \frac{1}{n^c}.$$

The negation of this definition is

$$\exists c \in \mathbb{N} : \forall n_0 : \exists n \geq n_0 : f(n) > \frac{1}{n^c},$$

which is not the same as a noticeable function (the function can be lower bounded by the inverse of some polynomial):

$$\exists c \in \mathbb{N} : \exists n_0 : \forall n \geq n_0 : f(n) \geq \frac{1}{n^c}.$$

It is not hard to obtain a function that is both not negligible and not noticeable. For example, we can take a noticeable function and a negligible function and interleave them. Let

$$f(n) = \begin{cases} 2^{-n}, & \text{for } n \text{ even} \\ n^{-2}, & \text{for } n \text{ odd} \end{cases}$$

Such a function is not negligible, because for odd integers, the function is only polynomially small. It is also not noticeable, because for even numbers, it is exponentially small.

c) A non-deterministic Turing machine can simply non-deterministically run the verifier on all possible witness strings. Observe that this requires only polynomially many steps, because it can non-deterministically choose the next character in the proof string in each step, and the length of the proof string is required to be polynomially bounded. If any proof is valid, some path will accept and the non-determinism will choose this path. If no proof is valid, the string is not in the language and it will reject.

Observe that the converse is also true. Suppose there is a non-deterministic Turing machine accepting a given language L. This means that there must be at least one accepting path, where it starts at some state S, travels through polynomially many states, and ends in a halting state H: $(S, \ldots, H)$. The polynomially long string describing this path is the witness supplied to the verifier. The verifier can then deterministically simulate the Turing machine, following only the accepting path, and verifying that it accepts at the end. If the Turing machine rejects the input, there is no accepting path, and the verifier will always reject.

2.2 Identification Protocols

a) Eve can eavesdrop the key k from the insecure channel and impersonate Alice by sending the key k to Bob.

b) Eve cannot compute the key k from the pair (v, c), but she can execute a replay attack as follows: Eve stores (v, c) and at a later time impersonates Alice by sending (v, c) to Bob. Bob will verify that c is the encryption of v with key k and identify Eve as Alice.

c) Alice and Bob can use the following protocol:

1. Bob chooses a random v, encrypts v with k, and sends the cyphertext c to Alice.
2. Alice decrypts the cyphertext, and sends v to Bob.

In this case Eve cannot reply the previous attack, since she cannot decrypt the cyphertext that Bob sends to Alice.

d) Let $m = pq$ be an RSA modulus (which may, e.g., have been generated by a trusted third party). At the conference meeting, Alice chooses a random $x \in \mathbb{Z}_m^*$ (her secret key), computes $z := x^2$ in $\mathbb{Z}_m^*$ (her public key), and hands z to Bob.

For Alice to identify herself to Bob, the Fiat-Shamir protocol is invoked, where Alice proves to Bob that she knows x. The protocol is invoked many times and Bob accepts if and only if he accepts in all invocations.

1. The completeness and soundness properties of the Fiat-Shamir protocol guarantee that Alice always succeeds in authenticating herself, whereas Eve (almost) always fails to do so.
2. Due to the zero-knowledge property of the Fiat-Shamir protocol, the messages sent to Bob do not reveal any information beyond the fact that Alice knows her secret key. This means that Eve also does not get any information.
3. Advantages: With the Fiat-Shamir protocol, Eve cannot use the transcript of the interaction at a later time to prove to anybody that Alice tried to authenticate herself, since she could have generated it herself. Moreover, if one accepts a cheating probability of $\leq 2^{-80}$, then the parties need to perform 80 multiplications each, which is much less than the number of multiplication typically needed to compute a signature.
   Disadvantage: Needs more communication and interaction as the Fiat-Shamir protocol needs to be repeated multiple times to get a low cheating probability.

2.3 Graph (Non-)Isomorphism

a) The GNI protocol from the lecture is not zero-knowledge because Vic could cheat by sending Peggy an arbitrary graph $\mathcal{K}$ and learn if $\mathcal{K}$ is isomorphic to $\mathcal{G}_0$ or $\mathcal{G}_1$.

b) The protocol is honest-verifier zero-knowledge, since in the case where the verifier follows the protocol, he chooses one bit b at random, and receives a bit $c = b$.

c) Let the three graphs be $(\mathcal{G}_0, \mathcal{G}_1, \mathcal{G}_2)$. The verifier permutes each graph randomly generating $\mathcal{H}_i$ with $\mathcal{H}_i \cong G_i$ for $i \in \{0, 1, 2\}$, and chooses a shift uniformly at random $s \in \{0, 1, 2\}$. Let $\mathcal{K}_i := \mathcal{H}_{(i+s) \bmod 3}$, for $i \in \{0, 1, 2\}$. The verifier sends (K_0, K_1, K_2)

to the prover, and the prover has to tell what s the verifier has chosen. If the prover succeeds, the verifier accepts. Otherwise, he rejects.

Completeness: If the three graphs $\mathcal{G}_0, \mathcal{G}_1, \mathcal{G}_2$ are not all isomorphic, the prover can tell which shift s the verifier has chosen.

Soundness: Assume that all graphs are isomorphic. In this case, the prover cannot tell which shift s the verifier has chosen. Hence, he cannot succeed with probability higher than one third.

Honest-Verifier Zero-Knowledge: Intuitively, the above protocol is honest-verifier zero-knowledge because a cheating verifier chooses a random shift s, and then receives $s' = s$ from the prover.