Solution Sheet #1

Advanced Cryptography 2021

Solution 1 Factorization

1. Each user needs 2 primes, thus, one needs a total of $2n$ prime numbers.

2. By taking the greatest common divisor of all possible pairs of moduli, Eve will be able to factorize all moduli for which at least one prime factor has been used in at least one other modulus.

3. Let $N_u = pq$ be the modulus of u. The probability that Eve can factorize N_u is the probability that there exist another modulus using *exactly* one prime from p, q (if the moduli are equal, the gcd will return N_u). The probability that another modulus cannot be used to factorize N_u is
$$\frac{\binom{k-2}{2}+1}{\binom{k}{2}}.$$
Hence, the probability that an adversary can factorize N_u is
$$1-\left(\frac{\binom{k-2}{2}+1}{\binom{k}{2}}\right)^{n-1}.$$

Solution 2 Square Roots

1. 1: **repeat**
   2: Pick a random $y_0 \in \{1, \ldots, n-1\}$.
   3: compute $x \coloneqq y_0^2 \bmod n$.
   4: Get $y \leftarrow \mathcal{O}(x)$.
   5: **until** $y \neq y_0$ and $y \neq -y_0 \bmod n$
   6: **return** $\gcd(y - y_0, n)$ which is one of the factors.

2. There are two square roots $\pm y_p$ of $x \bmod p$ and two square roots $\pm y_q$ of $x \bmod q$. The four square roots of $x \bmod n$ correspond to $\mathrm{CRT}(\pm y_p, \pm y_q)$ for all sign combinations.

   If $y \neq y_0$ and $y \neq -y_0$, this means that y and y_0 have a different square root mod p or mod q (it's an exclusive or). Let's say without loss of generality that they differ mod p. Then, $y \neq y_0 \bmod p$ and $y = y_0 \bmod q$. Hence, $y - y_0 \neq 0 \bmod p$ and $y - y_0 = 0 \bmod q$. We have, thus, that $y - y_0$ is a factor of q and not of p and we get q through the gcd.

3. They are four square roots mod n and only two of them gives us a factor (the other two are either y_0 or $-y_0$). Hence, we need two queries in average. The complexity of the algorithm is $O((\log(n))^2 + |\mathrm{SQRT}|)$, where $|\mathrm{SQRT}|$ is the complexity of the oracle.

Solution 3 Complexity

1. $f(n) = \Omega(g(n))$ is equivalent to $\exists c \in \mathbb{R}^+ \; \exists n_0 \in \mathbb{N} \; \forall n \in \mathbb{N} \quad n \geq n_0 \implies |f(n)| \geq c \cdot |g(n)|$.
   $f(n) = \Theta(g(n))$ is equivalent to $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$, alternatively

   $$\exists c_0, c_1 \in \mathbb{R}^+ \; \exists n_0 \in \mathbb{N} \; \forall n \in \mathbb{N} \quad n \geq n_0 \implies c_0 \cdot |g(n)| \geq |f(n)| \geq c_1 \cdot |g(n)|.$$

   $f(n) = o(g(n))$ is equivalent to $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$, alternatively

   $$\forall \epsilon \in \mathbb{R}^+ \; \exists n_0 \in \mathbb{N} \; \forall n \in \mathbb{N} \quad n \geq n_0 \implies \left|\frac{f(n)}{g(n)}\right| < \epsilon.$$

2. From the previous definition, we need to provide a tuple (c_0, c_1, n_0) such that the statement holds. Since $-1 \leq \sin n \leq 1$, for a choice of $c_0 = 3$, $c_1 = 1$ and $n_0 = 1$, we can deduce that $n \leq n(2 + \sin n) \leq 3n$ for all $n \geq 1$.

Solution 4 Negligible function

In order to check if these functions are negligible or not we have to verify the definition. If f is negligible, we can say how to choose n_0 such that for every c the definition is satisfied. Otherwise, we give a counter example, i.e. valid values for c and n_0 such that the inequality is not respected.

1. $f(x) = 2^{-x}$ is negligible. We can choose n_0 such that $\frac{n_0}{\log_2 n_0} > c$ such that the definition is verified.

2. $f(x) = x^{-3}$ is not negligible. The function is only polynomially small and we can pick $n_0 = 1$ and $c = 3$ to invalidate the definition.

3. $f(x) = 100 \cdot x^{-x}$ is negligible. We can choose n_0 such that $(n_0 - c) \log_2 n_0 > log_2(100)$ such that the definition is verified.

4. $f(n) = \begin{cases} 5^{-x} & \text{if } x \text{ is even} \\ x^{-10000} & \text{if } x \text{ is odd} \end{cases}$ is not negligible. For x values that are odd, the function is only polynomially small and we can choose $n_0 = 1$ and $c = 10000$.

Solution 5 Formalism

1. A pseudorandom number generator is a tuple $(\{0,1\}^k, \{0,1\}^\tau, PRNG)$ such that

   - $\{0,1\}^k$ is the state domain and $k(s)$ is polynomially bounded
   - $\{0,1\}^\tau$ is the output domain and $\tau(s)$ is polynomially bounded
   - $PRNG$ is a deterministic polynomially bounded algorithm such that

     $$PRNG : \{0,1\}^k \mapsto \{0,1\}^k \times \{0,1\}^\tau.$$

2. We define success probability function[1] $Adv^{sr}_{\mathcal{A},PRNG}$ of $\mathcal{A}$ in *state recovery* game as follows:

$$Adv^{sr}_{\mathcal{A},PRNG}(s,d) = \Pr_{state_0,\zeta}[\mathcal{A}(r_1,r_2,\ldots,r_d;\zeta) = state_d]$$

where $state_0$ is uniformly drawn from $\{0,1\}^k$ and others are generated with $(state_{i+1}, r_{i+1}) = PRNG(state_i)$ for $i = 0,1,\ldots,d-1$ and ζ is the random coins provided to the adversary.

Similarly, for *indistinguishability*, we define the success probability as follows:

$$Adv^{ind}_{\mathcal{A},PRNG}(s,d) = \Pr_{state_0,\zeta}[\mathcal{A}(r_1,r_2,\ldots,r_d;\zeta) = 1|exp_0] - \Pr_{r_1,\ldots,r_d,\zeta}[\mathcal{A}(r_1,r_2,\ldots,r_d;\zeta) = 1|exp_1]$$

where exp_i events define two different setups of parameters:

- For exp_0, $state_0$ is uniformly drawn from $\{0,1\}^k$, and $(state_{i+1}, r_{i+1}) = PRNG(state_i)$ for $i = 0,1,\ldots,d-1$ (just as in state recovery game).
- For exp_1, r_i values are uniformly drawn from $\{0,1\}^\tau$ for $i = 1,\ldots,d$.

3. If an efficient state recovering adversary $\mathcal{A}$ exists with success probability $Adv^{sr}_{\mathcal{A},PRNG}$ for a given $PRNG$, then we can construct a distinguisher $\mathcal{B}$ who uses $\mathcal{A}$ as a subroutine defined as follows:
   (a) receive $r_1,\ldots,r_d$ values from the experiment
   (b) run $\mathcal{A}(r_1,\ldots,r_{d-1}) \to (state)$
   (c) run $PRNG(state) \to (state_{next}, r'_d)$
   (d) return $1_{r_d = r'_d}$ (i.e. 1 iff $r_d = r'_d$)

   The success probability of $\mathcal{B}$ is defined as:

$$|\Pr[\mathcal{B} = 1|exp_0] - \Pr[\mathcal{B} = 1|exp_1]|$$

   where exp_0 and exp_1 experiments are defined above.

   We notice that $\Pr[\mathcal{B} \to 1|exp_0]$ is lower bounded by the probability of $\mathcal{A}$'s winning the state recovery game with $d-1$ samples, because if $\mathcal{A}$ managed to guess the correct state, $PRNG(state)$ would definetly give the next output.

   On the other hand, $\Pr[\mathcal{B} \to 1|exp_1] = 2^{-\tau}$ (which is negligible in terms of the implicit security parameter s). Therefore, the probability of $\mathcal{B}$'s success in distinguishing game is:

$$Adv^{ind}_{\mathcal{B},PRNG} = \Pr[\mathcal{B} \to 1|exp_0] - \Pr[\mathcal{B} \to 1|exp_1] \geq Adv^{sr}_{\mathcal{A},PRNG} - 2^{-\tau}.$$

   It follows that if $Adv^{sr}_{\mathcal{A},PRNG}$ is not negligible, then $Adv^{ind}_{\mathcal{B},PRNG}$ is also not negligible.

$$\exists \text{state recovery attack} \Rightarrow \exists \text{distinguisher}$$

   which is equivalent of

$$\nexists \text{distinguisher} \Rightarrow \nexists \text{state recovery attack}.$$

[1] Oh, wow, such an ugly notation! Success probabilities are usually called advantage and one school of thought follows the convention of denoting them by a cumbersome but useful notation: $Adv^{notion}_{\mathcal{A},Prm}(params)$. The excuse for drowning in such garbled mess of miscellaneous variables is that technically, the associated probability function depends on plenty of things: the security game definition *notion*, an adversary $\mathcal{A}$, a primitive tuple Prm consisting of domains and algorithms directly used by the experiment; and some security parameters *param*. We could just assign variable anew for each new probability, e.g. p, but one mostly winds up having to deal with a number of notions and algorithms; therefore adapting to a standard convention is wise in hindsight.