Problems for week 6, Cryptography Course - TDA 352/DIT 250

General remarks on problems for the weekly problem session: Exercises will be classified in four different levels:

1. **Easy**: the exercise will require low numerical computations or it can be just a way to look back at the content of the lecture. Exercises of this level should easily be done with just *pen and paper* and are **important to pass** the exam.
2. **Medium**: the exercises will require some time to do (from 5 to 15 minutes each). Maybe a separate paper for some computation is needed! You need to study a bit to answer the questions. These exercises also **may appear** in the exam.
3. **Hard**: the exercises will require you to spend a lot of time doing numerical computations (and we highly recommend using a PC) **or** the questions are real challenge to see if you understood the course in depth. Some of these exercises **may appear** in the exam.
4. **Think**: problems that aim to using your imagination. You are invited to discussion with your colleagues/friends/family and find your best solutions. Generally, the exercises of this level do not take a lot of time in writing the solutions but they will let you think/discuss for (maybe) 30/40 minutes.

Easy

---

1. A hash-then-sign scheme to avoid Multiplication forgery is RSA-PPS. A simplified version of the PSS scheme is described below. Please note that the scheme uses a Mask Generation Function (G), this function has similar properties like hash functions, but it produces a variable length output.

   To sign message m, Alice proceeds as follows:

   - Choose random r and compute $w = H(m||r)$.
   - Compute $G_0(w) = g_0$.
   - Compute $G_1(w) = g_1$.
   - put $y = 0||w||g_0 \oplus r||g_1$, where 0 is padding
   - Compute the RSA signature s on y, i.e. $s = y^d \bmod N$.

   To verify the signature s' on messsage m, Bob proceeds as follows.

   - Compute $y' = s'^e \bmod N$.
   - Parse y' as $y' = b||w'||\alpha||\gamma$.
   - Compute $G_0(w') = g'_0$.
   - Compute $G_1(w') = g'_1$.
   - Compute $r' = \alpha \oplus g'_0$.
   - Check that $b = 0$, $g'_1 = \gamma$ and $H(m||r') = w'$.

   Verify that the method described above for verifying RSA-PSS signatures is correct.

2. Consider the following signature scheme. The scheme group is $\mathbb{Z}_q$ for a large prime q and a generator g of $\mathbb{Z}_q$. A user has a private key α and a public key $X = g^\alpha$.

   To sign a message m, one first computes $h = H(m)$ for some hash function H. Then, one computes $z = \alpha/h$ (assuming $h \neq 0$). The signature is g^z.

   The verification of the signature s consists of checking that $s^h = X$. Is this a good scheme, i.e.

   (a) Will correct signatures be accepted?

   (b) Is it infeasible to sign an arbitrary message without knowing α?

3. Alice uses the RSA signature scheme and publishes the following data : $n = pq = 221$ and $e = 13$. Bob receives the message $P = 65$ and the corresponding digital signature $S = 182$.Verify the signature.

4. Let $p = 467$ and $g = 2$. Alice uses the ElGamal signature scheme, and her public key is $y = 132$.

   a. Verify that $(29, 51)$ is a valid signature of Alice for the message $m = 100$.

   b. Suppose your secret key is $x = 127$. Sign the message $m = 110$.

   Hint: Feel free to use calculators when needed !

5. Suppose that Bob uses the DSA with $q = 103$, $p = 10q + 1 = 1031$, $g = 275$, $x = 75$, and $y = 339$. Determine Bob's signature on the message $m = 908$ using the random value $k = 49$, and verify the resulting signature.
   Hint: Feel free to use calculators when needed !

Medium

---

6. State and prove the birthday paradox.

7. An attack that can be done against hash functions is to pre-compute a **rainbow table** that consist of a table with the pair: message and the hash value for that message. Generally these tables are optimized for research and retrieve the *inverse* for the hash function. You can easily find online rainbow tables for passwords! We will consider SHA256 as the hash function and we want to compute the dimension of a rainbow table for a fixed length password:
   suppose that we have a password of length $n = 5$. SHA256 generates a 256-bit digest (32 byte) and a single n-character long password can be represented with $n \cdot 8 = 40$-bit[1]. The number of characters that can be used for the password is 64 (26 lowercase letters, 26 uppercase letters, 10 digits and 2 symbols like *':'* and *'.'*)[2].

   a. How much is the size of the rainbow table in this case where the password length is $n = 5$?

   Recently, NIST opened to the public their new draft for security procedures and in this draft there is a clear definition on *good-practices* for generating a secure password.
   In this draft, NIST suggested minimal length for password password should be $n = 8$ character long.

   b. How much is the size of the rainbow table in this case with $n = 8$ with 64 possible different characters?

   Another suggestion is to avoid rules like *"your password must contains at least a symbol, two digit, 3 lowercase letter, a uppercase letter but not in a even position and the length must be equal to 2 modulo 3"*[3] in order to just use the alpha-numeric character-set (so 62 different characters).

   c. How much is the size of the rainbow table in this case with $n = 8$ with 62 possible different characters?

   d. What is your opinion/idea on password rules in order to achieve a secure password?

8. Let $H : M \to T$ be a collision resistant hash function. Which of the following is/are collision resistant?

   (a) $H'(m) = H(0)$

   (b) $H'(m) = H(m\|m)$ ($\|$ denotes the concatenation)

   (c) $H'(m) = H(m) \oplus H(m)$

   (d) $H'(m) = H(m) \oplus H(m \oplus 1^{|m|})$

   (e) $H'(m) = H(m)\|H(0)$

   (f) $H'(m) = H(H(m))$

[1] We suppose you use ASCII 8-bit representation for a single char.
[2] Usually the number of symbols is 10 but it really depends on the implementation.
[3] Impossible password rules exists.

9. Suppose H_1 and H_2 are collision resistant hash functions mapping inputs in a set M to $\{0,1\}^{256}$. Show that the function $H_2(H_1(m))$ is also collision resistant.
Hint: Prove the contra-positive.

10. Consider the following proposal for a cryptographic hash function, that makes use of a block cipher E with block size k bits to produce k bit hash values.

    The message to be hashed is split into a sequence $M_1 M_2 ... M_n$ of k bit blocks. For simplicity, we ignore padding and consider only messages whose length is a multiple of the block size. Hashing works as follows:

    $$h_0 = IV$$
    $$h_i = E_{M_i}(h_{i-1}) \text{ for } i = 1, 2, ..., n,$$

    The hash of the message is h_n. Here IV is a fixed, public initialisation vector, which is part of the definition of the hash function.

    The purpose of this problem is to show that this is not a very good scheme. Given *any* hash value h and *any* message M, one can design a meet-in-the-middle attack that extends M with two blocks to produce a message that has h as hash value. The expected running time of the attack is in the order of $2^{k/2}$ encryptions and decryptions of a single block. Describe such an attack in detail, including motivations for time and space requirements.

    To understand the adversary's advantage better: If he has Alice's signature on *just one* message (assuming that she signed the hash of the message) this problem shows that he also has Alice's signature on $M\|X\|Y$ where M is *any* message that the adversary chooses and X and Y are two garbage blocks that he has to add to get the signature to work. The garbage at the end is of course a nuisance to the adversary, but this property is anyhow unacceptable and the construction must be rejected as a hash function.

11. We consider the possibility of using SHA-1 or MD5 for authentication as follows. Bob authenticates message m for Alice by computing $h(K\|m\|p)$ where h is the hash function, K is the secret key shared between Alice and Bob, and p is padding. Demonstrate that this system has the (unwanted) property that the Adversary can authenticate certain messages not sent by Bob.

Hard

12. Let $h : \{0,1\}^* \longrightarrow \{0,1\}^n$ be a hash function that is second-preimage and collision resistant. Let $h' : \{0,1\}^* \longrightarrow \{0,1\}^{n+1}$ be the hash function given by the rule

    $$h'(x) = \begin{cases} 0\|x & x \in \{0,1\}^n \\ 1\|h(x) & \text{otherwise.} \end{cases}$$

    Prove that h' is not preimage resistant, but still second-preimage and collision resistant.

13. In this problem we consider the ElGamal signature scheme which works as follows:

    The setting is $\mathbb{Z}_p^*$ for a large prime p and a generator g for $\mathbb{Z}_p^*$. g and p can be common parameters for a community of users. The scheme also makes use of a suitable cryptographic hash function h.

    Each user chooses a private long-term key $x \in \mathbb{Z}_{p-1}$ and computes his public key $X = g^x$ in $\mathbb{Z}_p^*$.

    To sign a message m, the following steps are performed:

    1. Choose a random y with $\gcd(y, p-1) = 1$.
    2. Compute $Y = g^y$ in $\mathbb{Z}_p^*$.
    3. Compute $z = y^{-1} \bmod (p-1)$.
    4. Compute $s = z \cdot (h(m) - xY) \bmod (p-1)$.

5. The signature is (Y, s).

To verify signature (Y, s) on message m, signed by a user with public key X, the following steps are performed:

1. Check that $1 \leq Y \leq p - 1$.
2. Check that $X^Y Y^s = g^{h(m)}$ in $\mathbb{Z}_p^*$.

Accept the signature if both checks succeed, reject it otherwise.

(a) Show that a correct signature will be successfully verified.

(b) Show that if a user signs two different documents using the same y, then an adversary can with high probability find out the user's private key x.

Think

14. Try to give a formal definition of a cryptographic hash function.
Two fundamental properties for cryptographic hash functions are the **collision resistance** (find two message with the same hash) and the *first-image resistance* (from a hash, find the original message). We consider a hash function **secure** if the complexity of *breaking* one of the properties is computationally unfeasible.
Is it possible to have an *"ideal perfect"* hash function that does not have any collision, i.e. all the messages have a unique hash value?
Hint: *try to think on the dimension of the message space and the digest space.*