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

Easy

1. The task is to demonstrate that a correct signature will be verified. We reason as follows.
   - Because of properties of RSA signatures, y' computed by Bob will equal y computed by Alice.
   - Hence, looking at how y was constructed and y' parsed, we will have $b = 0$, $w' = w$, $\alpha = g_0 \oplus r$ and $\gamma = g_1$.
   - Since G was applied to the same argument ($w = w'$), we have that $g'_0 = g_0$ and $g'_1 = g_1$.
   - Hence $r' = \alpha \oplus g'_0 = g_0 \oplus r \oplus g_0 = r$.
   - We already noted that $b = 0$. We also get from the previous steps that $H(m||r') = H(m||r) = w'$ and $g'_1 = g_1 = \gamma$. Hence the verification will succeed.

2. (a) Correct signatures will be accepted, since
$$s^h = g^{zh} = g^\alpha = X$$
   (b) **NO!** We have from the above equation that
$$s = X^{h^{-1}}$$
   Given a message, anyone can hash it to get h, compute h^{-1} by the Extended Euclidean algorithm and then compute the signature using the user's public key.

3. The signature is valid if
$$P = S^e \bmod n.$$
   In our case:
$$S^e \bmod n = 18213 \bmod 221 = 65,$$
   which is valid.

4. a. It suffices to show that $g^m = y^r r^s \bmod p$. Indeed,
$$y^r r^s \bmod p = 132^{29} \cdot 29^{51} \bmod 467 = 189$$
   and
$$g^m = 2^{100} \bmod 467 = 189$$
   b. $y = g^x \bmod p = 2^{127} \bmod 467 = 132$. Then, my pair of keys is:$(127, 132)$. Now, pick $k = 213$. Check that $\gcd(213, 466) = 1$. It holds that
$$213^{-1} \bmod 466 = 431.$$
   Thus,
$$r = g^x \bmod p = 2^{127} \bmod 467 = 29$$
   and
$$s = (m - xr)k^{-1} \bmod p = (110 - 127 \cdot 29) \cdot 431 \bmod 467 = 203.$$
   Finally,
$$(r, s) = (29, 203)$$

5.

$$\begin{aligned} r &= (g^k \bmod p) \bmod q \\ &= (275^{49} \bmod 1031)) \bmod 103 \\ &= 552 \bmod 103 \\ &= 37 \end{aligned}$$

and

$$\begin{aligned} s &= k^{-1}(m + xr) \bmod q \\ &= (82 \cdot (908 + 75 \cdot 37)) \bmod 103 \\ &= (82 \cdot 3683) \bmod 103 \\ &= 10 \end{aligned}$$

Thus, Bob's signature is $(r, s) = (37, 10)$

Now, we will verify his signature.

$$\begin{aligned} w &= s^{-1} \bmod q \\ &= 10^{-1} \bmod 103 \\ &= 31 \end{aligned}$$

and

$$\begin{aligned} u_1 &= m \cdot w \bmod q \\ &= 908 \cdot 31 \bmod 103 \\ &= 29 \end{aligned}$$

and

$$\begin{aligned} u_2 &= r \cdot w \bmod q \\ &= 37 \cdot 31 \bmod 103 \\ &= 14 \end{aligned}$$

Now we can calculate v :

$$\begin{aligned} v &= (g^{u_1} \cdot y^{u_2} \bmod p) \bmod q \\ &= (275^{29} \cdot 339^{14} \bmod 1031) \bmod 103 \\ &= 552 \bmod 103 \\ &= 37 \end{aligned}$$

Since $v = r = 37$ we are done.

Medium

6. **Proposition:**[1] Given a year with N days, the generalized birthday problem asks for the minimal number n such that, in a set of n randomly chosen people, the probability of a birthday coincidence is at least 50% (assuming that birthdays are independent random variables with same distribution $\Pr(X = \text{ day }) = \frac{1}{N}$.
In other words, n is the minimal integer such that

$$1 - \prod_{i=1}^{n-1} \left(\frac{N-i}{N} \right) \geq \frac{1}{2}$$

[1] Link to a interactive explanation of the birthday problem.

and

$$n \simeq 1.177\sqrt{N}$$

Proof:

Fact 1: If $x_1, \cdots, x_n$ are real numbers, then

$$\prod_{i=1}^{n}(1-x_i) \leq e^{-\sum_{i=1}^{n} x_i}$$

Fact 2: for a natural number n

$$\sum_{i=1}^{n} i = \frac{1}{2}n(n+1)$$

From these two facts, we obtain

$$\prod_{i=1}^{n-1}\left(1-\frac{i}{N}\right) \leq e^{-\sum_{i=1}^{n-1}\frac{i}{N}} = e^{-\frac{1}{N}\sum_{i=1}^{n-1} i} = e^{-\frac{(n-1)n}{2N}}$$

We consider now

$$e^{-\frac{(n-1)n}{2N}} \geq \frac{1}{2}$$

from which we can compute n:

$$\begin{aligned} e^{-\frac{(n-1)n}{2N}} &\geq \frac{1}{2} \\ \ln\left(e^{-\frac{(n-1)n}{2N}}\right) &\geq \ln\left(\frac{1}{2}\right) \\ -\frac{(n-1)n}{2N} &\geq -\ln 2 \\ \frac{(n^2-n)}{2N} &\leq \ln 2 \\ (n^2-n) &\leq \ln 2 \cdot (2N) \end{aligned}$$

Since we are considering large n values, we can consider $n^2 - n$ to be equivalent to n^2 and so:

$$\begin{aligned} n^2 &\leq \ln 2 \cdot (2N) \\ n_{1,2} &= \frac{\pm\sqrt{4 \cdot \ln 2 \cdot (2N)}}{2} \\ &\text{just consider the solution with positive sign since } n \geq 0 \\ n &= \frac{\sqrt{8\ln(2)}}{2} \cdot \sqrt{N} = \sqrt{2\ln(2)} \cdot \sqrt{N} \simeq 1.177\sqrt{N} \end{aligned}$$

7. To solve this exercise, we study a more abstract scenario:
we have k symbols in an alphabet (language, encoding, a symbolic representation) that will be encoded with b-bit per symbol. Let n be a fixed length of a *word* that is represented with a index of b_w bits (a single b_w-bit string representation to uniquely define a single word). We can now compute the *"number of bit necessary to storage all the n-symbol long word and their unique representation"* where b_n is the length of the representation and b is the number of bit necessary to represent a word per symbol.

$$\text{Size} = (k^n)(b_n + n \cdot b) =$$

$=$ (number of possible words) $\cdot$ (bit necessary to represent a word and its unique representation)

With this general formula, we can see the results in our specific cases:

a. We have $n = 5$ length string of $k = 64$ represented with $b = 8$ bit per symbol. SHA256 has a digest of 256 bit and so $b_w = 256$.
So, applying the general formula, we obtain:

$$\text{Size} = (k^n)(b_n + n \cdot b) = (64^5)(256 + 5 \cdot 8) \simeq 2^{30}2^8 = 2^{38} \text{ bits} \simeq 34.3 \text{ GigaBytes}$$

b. In this case, we changed the length to be $n = 8$. In the formula:

$$\text{Size} = (k^n)(b_n + n \cdot b) = (64^8)(256 + 8 \cdot 8) \simeq 2^{48}2^8 = 2^{56} \text{ bits} \simeq 9 \text{ Peta-Bytes}$$

c. We have the same values as the point before, but we have $k = 62$.

$$\text{Size} = (k^n)(b_n + n \cdot b) = (62^8)(256 + 8 \cdot 8) \simeq 2^{47}2^8 = 2^{55} \text{ bits} \simeq 4.5 \text{ Peta-Bytes}$$

d. As we can see by the formula, the strength of a password with respect a rainbow-table attack depends mostly on the number of symbols allowed (freely without strange rules) and the length of the password.
For this reason, long password should be preferred.
On the other hand, a secure password can become weak if the implementations, the protocols and schemes that uses that password are not secure.

8. (a) No. For any m, m' such that $m \neq m'$ it holds:

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

(b) Yes. Suppose there exists m, m' such that $m \neq m'$ and $H'(m) = H'(m')$, then:

$$H'(m) = H'(m') \Leftrightarrow H(m\|m) = H(m'\|m')$$

where, of course, $m\|m \neq m'\|m'$. We found a collision for H. Contradiction!

(c) No. For any m, m' such that $m \neq m'$ it holds:

$$H'(m) = H(m) \oplus H(m) = 0^{|m|} = 0^{|m'|} = H(m') \oplus H(m') = H'(m')$$

(d) No. Let $m = 0^{|l|} \neq m' = 1^{|l|}$. Here, of course, $|m| = |m'| = l$. Then, it holds:

$$H'(0^{|l|}) = H(0^{|l|}) \oplus H(0^{|l|} \oplus 1^{|l|}) = H(0^{|l|}) \oplus H(1^{|l|}) = H(1^{|l|} \oplus 1^{|l|}) \oplus H(1^{|l|}) = H'(1^{|l|})$$

(e) Yes. Suppose there exists m, m' such that $m \neq m'$ and $H'(m) = H'(m')$, then:

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

We found a collision for H. Contradiction!

(f) Yes. We know that H is collision resistant. Thus, for any m, m' such that $m \neq m'$ it holds that $H(m) \neq H(m')$. For the same reason, $H(H(m)) \neq H(H(m'))$ as well. That is, $H'(m) \neq H'(m')$.

9. Suppose $H_2(H_1(\cdot))$ is not collision resistant, that is, we are given $x \neq y$ such that $H_2(H_1(x)) = H_2(H_1(y))$. We will build a collision for either H_1 or for H_2. This will prove that if H_1 and H_2 are collision resistant then so is $H_2(H_1(\cdot))$.
Let $x \neq y$ and $H_2(H_1(x)) = H_2(H_1(y))$. We have two possible cases:

i. $H_1(x) = H_1(y)$ Since $x \neq y$ it gives a collision on H_1.

ii. $H_1(x) \neq H_1(y)$ Since $H_2(H_1(x)) = H_2(H_1(y))$ it gives a collision on H_2.

10. As the first step we hash M and get, say, h_0. We want to produce a message of the form $M\|X\|Y$ with hash value h. To this end, we choose $2^{k/2}$ random blocks and for each such block X we compute $h_X = E_X(h_0)$. We store the pairs (h_X, X) in a hash table, indexed by h_X. Then we start from the end, trying to meet in the middle: We choose again random blocks and for each such block Y we compute $h_Y = E_Y^{-1}(h)$ and try to look up h_Y in the hash table. If we find it, we have found the suitable X and Y and the attack has succeeded. The birthday paradox tell us that the number of blocks Y that we have to try is in the order of $2^{k/2}$.

Time complexity is, as stated, around $2^{k/2}$ steps, where a step includes one encryption, one hash table insertion, one decryption and one hash table lookup. Space requirements for the hash table is $2^{k/2}$ pairs, where each pair is $2k$ bits.

11. Assume that Bob has authenticated message m by computing the hash value for $K\|m\|p$ as h. Then the Adversary can compute the message digest for $K\|m\|p\|m'\|p'$ where m' is any message and p' is the padding needed for the extended message. He can do this since the hash functions work by splitting the message into blocks $B_1, B_2, ..., B_n$ and iterating:

$$H_0 = IV$$
$$H_k = g(B_k, H_{k-1}) \text{ for } k = 1, 2, ..., n,$$

where g is the compression function. Thus the Adversary can compute the hash value for the extended message by starting from h and iterating over the blocks he added.

Note that the message authenticated in this way is $m\|p\|m'$, i.e. it extends message m by m' but it also contains the padding p, which might be a nuisance to the Adversary.

Hard

---

12. The modified hash function h' is not preimage resistant, since for any hash value y of the form $0\|x$, a preimage is x.

    Therefore, we can find a preimage for at least one half of all possible hash values.

    Next we prove that h' inherits second-preimage and collision resistance from h. We show that if we can find a collision or a second preimage for h', then we can easily do so for h. Suppose

    $$\exists x_0 \neq x_1 : h'(x_0) = h'(x_1).$$

    Two cases:

    (a) First bit of $h'(x_0)$ is 0. Impossible as implies $x_0 = x_1$.

    (b) First bit of $h'(x_0)$ is 1. Then, $h(x_0) = h(x_1)$ a contradiction, as h is collision resistant.

13. (a) The first check will succeed: since Y is computed in $\mathbb{Z}_p^*$ it will lie in the stated interval. For the second, we proceed as follows. Steps 3 and 4 in the signing procedure gives $ys = h(m) - xY$ modulo $p-1$ and thus

    $$(g^x)^Y (g^y)^s = g^{xY+ys} = g^{h(m)} \bmod p$$

    where Fermat's little theorem justifies replacing $xY + ys$ by $h(m)$ in the exponent since these quantities are equal modulo $p-1$.

    Note: One could wonder whether the first check, that $1 \leq Y \leq p-1$, is important. It actually is, as the following attack shows. This is of course not part of the problem as stated.

    Assume that (Y, s) is a signature on message m and that receivers omit check 1. The adversary can then create a signature on an arbitrary message m' as follows:

    1. Compute $u = h(m') \cdot h(m)^{-1} \bmod (p-1)$.
    2. Compute $s' = su \bmod (p-1)$
    3. Compute Y' that satisfies $Yu = Y' \bmod (p-1)$ and $Y = Y' \bmod p$, using the Chinese Remainder Theorem.
    4. A signature on m' is (Y', s').

    Note that Y' will be a number $< p(p-1)$ and thus not pass check 1. The proposed signature will, however, satisfy check 2 (we compute in $\mathbb{Z}_p^*$, so in the exponent we can compute modulo $p-1$):

    $$X^{Y'} Y'^{s'} = X^{Yu} Y^{su} = (X^Y Y^s)^u = g^{h(m)u} = g^{h(m')}$$

    (b) We assume that the adversary has signatures (Y, s_1) and (Y, s_2) on messages $m_1 \neq m_2$. He can then conclude from 4 in the signing process that

    $$s_1 = z \cdot (h(m_1) - xY) \bmod (p-1),$$

    $$s_2 = z \cdot (h(m_2) - xY) \bmod (p-1).$$

    Only z and x are unknown here; the goal is to compute x. He subtracts the two equations term-wise to get

    $$s_1 - s_2 = z \cdot (h(m_1) - h(m_2)) \bmod (p-1).$$

    We assume that $h(m_1) \neq h(m_2)$, which holds with probability close to 1. We consider two cases:

i. $\gcd(h(m_1) - h(m_2), p-1) = 1$. The adversary can then invert $h(m_1) - h(m_2)$ in $\mathbb{Z}^*_{p-1}$ and get

$$y^{-1} = z = (s_1 - s_2)(h(m_1) - h(m_2))^{-1} \bmod (p-1).$$

From this he can compute y using the extended Euclidean algorithm.

ii. $\gcd(h(m_1) - h(m_2), p-1) = d \neq 1$. In this case we get d solutions for z and hence d possible values for y. The adversary can check which of these is correct by computing g^y and comparing with Y . So, also in this case the adversary succeeds in finding y, if only d is small enough to make the procedure outlined feasible.

When the adversary has found out y, he needs to solve for x. Rewriting one of the equations above, he arrives at

$$Yx = (h(m_1) - ys_1) \bmod (p-1).$$

With similar reasoning as above he can now solve for x unless $\gcd(Y, p-1) = d'$ is so large that it is infeasible to check all potential solutions to find the correct one (this check is done by recomputing the signature). Summing up, we have shown that the adversary who has access to two messages and signatures signed with the same y can break the system unless the two messages hash to the same value or either the difference between the two hash values or Y has a common factor with $p-1$ that is so large that the computations above become infeasible. The probability for any of these cases is very small.

Think

14. **Definition:** Let $d \in \mathbb{N}$ a positive natural number. A function $f : \bigcup_{n=1}^{\infty} \{0,1\}^n \to \{0,1\}^d$ that takes as input an arbitrarily long bit-string and outputs a fixed length bit-string (called digest) is called **(cryptographic) hash function**.
A hash function f is called a **secure** cryptographic hash function is the following properties hold:

- f is **collision resistant** (i.e., it is unfeasible to find two messages m_1, m_2 such that $f(m_1) = f(m_2)$) and
- f is **first-image resistant** (i.e., it is unfeasible, given a digest y, to find the message m such that $f(m) = y$)

An *ideal perfect* hash function without collisions does not exists in reality.
The simplest way to see this is:
suppose that there exists a ideal perfect secure cryptographic hash function $f : \bigcup_{n=1}^{\infty} \{0,1\}^n \to \{0,1\}^d$ this means that f sends an infinite set into a finite set.
Being collision resistant can be translated as: *"the counter-image X of a digest y (you can think of X as the set of inverse $f^{-1}(y)$) is an unique value"* (i.e., $X = \{m\}$).
Otherwise, if X contains more than one element, say $X = \{m_1, m_2\} = f^{-1}(y)$ then it means that we found a collision, i.e., $f(m_2) = y = f(m_1)$.
So, to avoid the collisions, the cardinality of the digest space should be bigger or at least equal to the message space, which is never the case.
Thus in reality, there exists no real case ideal perfect secure cryptographic hash function.