Introduction to Mathematical Logic, Spring 2018
Homework 1

B04902012 Han-Sheng Liu, CSIE, NTU
E-mail: b04902012@ntu.edu.tw

1 CIA

1.1 Confidentiality

Prevent unarthorized person from accessing the information.
Example: Almost every account system requires login, to prevent the person who doesn't know the password from accessing the information.

1.2 Integrity

Assure the completeness of the information.
Example: When providing an software application, not only give the download link but also give the hash value of the file.

1.3 Avalibility

Maintain the correct function of a service.
Example: Use proof-of-work protocol to prevent part of denial-of-service attacks.

2 Hash Function

Let $H()$ be the considered hash function.

2.1 One-wayness

Given $H(x)$, it's hard to find x.
Example: In a database of account system, we often store $H(\text{password})$ instead password itself, to keep confidentiality even when the database is hacked. If it is easy to find H's preimage, then this method is no longer secure.

2.2 Weak Collision Resistance

Given x, it's hard to find $y \neq x$ such that $H(x) = H(y)$
Example: A hash value of the file is often provided with the file itself to assure integrity. If it's easy to find another file that has the same value as the original file, then a hacker (or even the provider himself) can modify the file, and the person who download may not be aware that the file is modified.

2.3 Strong Collision Resistance

It's hard to find $y \neq x$ such that $H(x) = H(y)$
Example: Google has found a pair of collision of SHA-1, due to some property of SHA-1, we can find infinity many pair of collision.

3 ElGamal Threshold Decryption

After choosing the large prime p, all the operation, such as addition, substraction, multiplication, division, and power, below will be done under $\mathbb{F}_p$.
`setup:`

$$\begin{aligned}
\text{large prime} &: p \\
\text{generator} &: g \\
\text{secret key} &: sk_B = b \\
\text{public key} &: pk_B = g^b \\
\text{secret random coefficient} &: a_1, a_2, ..., a_{t-1} \\
\text{polynomial} &: F(x) = b + \sum_{i=1}^{t-1} a_i x^i \\
\text{secret fragments} &: b_i = (i, F(i)), \text{ for all } i = 1, 2, ..., n
\end{aligned}$$

Distribute the n fragments to the n people. Let the people be p_1, p_2, p_n, give b_i to p_i

`encryption:`

$$\begin{aligned}
\text{plaintext} &: m \\
\text{random value} &: x \\
\text{ciphertext1} &: c_1 = g^x \\
\text{ciphertext2} &: c_2 = m(pk_B)^x
\end{aligned}$$

`key construction:`
W.L.O.G, let the collaborating t people be $p_1, p_2, ..., p_t$, and thus they have $b_1, b_2, ..., b_t$.

$$\text{secret key} : b = \sum_{i=1}^{t} F(i) \prod_{j \neq i} j(j-i)^{-1}$$

`decryption:`

$$\text{plaintext} : m = c_2 c_1^{-b}$$

4 How2Crypto

After several attemps, I collected some fragments of the plaintext. I googled those plaintext fragments, and got the original plain text. It is actually the first paragraph of `https://en.wikipedia.org/wiki/Cryptography`. By obtaining the original plaintext, all of the challenges became much more easier than before.

- *Round 1:* Parse the integer sequence into several two-digit integers, and convert them into english alphabets by their order in alphabets list.
- *Round 2:* It's encrypted by Caesar cipher, and the key can be obtained by observing `m1` and `c1`.
- *Round 3:* It's encrypted by Caesar cipher, consider all the possible key, and check if the decryption result is a substring in original plain text.
- *Round 4:* It's encrypted by substitution cipher. By observing `m1` and `c1`, we can decrypt part of the `c2`. Fill the part that is not decrypted with `'.'`, do a ReGex match on the original plaintext, and then send the match result.
- *Round 5:* It's encrypted by one of the permutation cipher. Enumerate all possible interval of the plain text, and check if the number of occurence of each alphabets meets `c2`.
- *Round 6:* The same as *round 5*.

After passing all the rounds, we will get 6 flag pieces. concat all of them, and decode it with base64 into a png file. The file is a picture with final flag on it.
`BALSN{You_Are_Crypto_Expert!!!^_^}`

5 Mersenne RSA

n is a 1128-bit number, so both p and q must be less than 2^1128. Since that both p and q are Merssene prime numbers, that is, a prime number of form $2^k - 1$, p and q have only at most 1128 possible values. Simply enumerate p and q as $2^k - 1$ for all $1 \leq k \leq 1128$, and check if $pq = n$. After finding out p and q, decrypt the flag by RSA decryption procedure.
`BALSN{if_N_is_factorized_you_get_the_private_key}`

6 OTP

Notation:

$$
\begin{aligned}
m &: \text{plain text} \\
c &: \text{cipher text} \\
m_i &: \text{The } i^{th} \text{ byte of plain text} \\
c_i &: \text{The } i^{th} \text{ byte of cipher text} \\
k &: \text{key} \\
k_i &: \text{The } i^{th} \text{ byte of the key} \\
l &: \text{length of the key}
\end{aligned}
$$

The plain text is printable, that is, $m_i < 128$ for all i. According to this property, if the number of bytes of the key is l, then $c_0 \oplus k_0, c_l \oplus k_0, c_{2l} \oplus k_0, ... < 128$, which implies that the first bit of $c_0, c_l, c_{2l}, ...$ must be the same. Generally, $c_i, c_{l+i}, c_{2l+i}, ...$ must also be the same for all $i < l$. Thus, by observing the period $P = 13$, we can get $13|l$. Let's assume that $l = 13$ first. If this assumption fails, we will assume that $l = 26$ secondly, and so on.

Now, it remains to calculate $k_0, ..., k_{12}$. Assume $c_0, ..., c_{12}$ be space (ASCII: 32) respectively, and we can get the corresponding $k'_0, ..., k'_{12}$. For $i < 13$, try decypting $c_i, c_{13+i}, c_{26+i}, ...$ with k'_i, and check if all of them are printable. If they are, then $k_i = k'_i$.

There are two reason we choose space.

- Space is the most common character in a English article, so it's more likely that assumption will be correct than choosing the other characters.
- The ASCII value of space is far from the ASCII values of other English characters, so if all decrypted characters are printable, then it's highly likely that the key is correct.

After the whole k is computed, decypt c with k and we will get the final flag.
`BALSN{NeVer_U5e_One_7ime_PAd_7wIcE}`

7 Double AES

Notation:

$$
\begin{aligned}
c &: \text{cipher text} \\
k_0 &: \text{The first key used for encryption} \\
k_1 &: \text{The second key used for encryption} \\
E_k(m) &: \text{Encrypt } m \text{ using AES with key } k \\
D_k(m) &: \text{Decrypt } m \text{ using AES with key } k
\end{aligned}
$$

In `2aes.txt`, a pair of plain and cipher text is provided. Let them be m_1 and c_1. Note that $c_1 = E_{k_1}(E_{k_0}(m_1))$ for some $k_0, k_1 < 2^23$. It is equivalent to $D_{k_1}(c_1) = E_{k_0}(m_1)$.

Compute $D_k(c_1)$ for all $k < 2^{23}$ and store all those $(D_k(c_1), k)$ pairs. Then, compute $E_k(m_1)$ for all $k < 2^{23}$, and check if there exists a k' such that $D_{k'}(c_1) = E_k(m_1)$. If so, then $k_0 = k$, $k_1 = k'$. The existence can be check in constant time in average by using hash table.

After get k_0 and k_1, we can get the final flag by computing $D_{k_0}(D_{k_1}(c))$.
`BALSN{so_2DES_is_not_used_today}`

8 Time Machine

Google provide a pair of pdf files with different contents but same $SHA1$ value on `https://shattered.io`. After truncate the same suffix, we get a pair of strings collides in $SHA1$ of length 2560 bits. Let the two strings be s_xi and s_y. By the property of *Merkle Damgard construction*, for all string s, $SHA1(s_x|s) = SHA1(s_y|s)$. That is, we can construct infinitely many pair of collision by appending the same string to s_x and s_y.

Randomly choose a string s repeatedly until the right most 24bits of $SHA1(s_x|s)$ meets the constraint. Then, send $SHA1(s_x|s)$ and $SHA1(s_y|s)$ to the server, and we get the final flkg.
`BALSN{POW_1s_4_w4st3_0f_t1m3_4nd_3n3rgy}`

9 Future Oracle

The procedure is as following.

1. Pick $ID =$ `admin`, $N_c = 7122$, *action*=`"login"`.
2. Send $(ID||N_c||SHA256(ID||N_c))$ to server.
3. Get N_s from server, let $Message = ID||N_s||$`login`.
4. Open a new connection, and send $ID||N_s||SHA256(ID||N_s)$ to server,
   get $Mac = SHA256(ID||N_s||$`login`$)$ from server,
   and then close the new connection.
5. Do length extension attack on Mac to get
   $newMessage = ID||N_s||$`login` <padding>||`printflag`,
   $newMac = SHA256(ID||N_s||$`login` <padding>||`printflag`$)$
6. Send $newMessage||newMac$ to server, and we will get the final flag.

`BALSN{Wh4t_1f_u_cou1d_s33_th3_futur3}`

10 Digital Saving Account

The procedure is as following. Let s^k be a character, say, `'_'`, repeat k times.

1. Pick username = s^4, pasword = s^1 and register. Let d_1 be the first 16 bytes of the decoded token.
2. Pick username = s^{10}`admin`, pasword = s^1 and register. Let d_2 be the last 32 bytes of the decoded token.
3. Pick token = $d_1|d_2$, username = s^4, password = s^1 and login.
4. Let t_1, t_2 be the string of those transactions, with (r_1, s_1), (r_2, s_2) be their signagure. Observe that r_1 always equals to r_2.
5. Compute $k = \dfrac{SHA1(t_1) - SHA1(t_2)}{s_1 - s_2} \mod q$.
6. Compute $s = \dfrac{k \times s1 - SHA1(t_1) + SHA1(\texttt{"FLAG"})}{k^{-1}} \mod q$.
7. Send r_1, s to the server, and we will get the final flag.

`BALSN{s3nd_m3_s0m3_b1tc01n_p13as3}`