18733: Applied Cryptography S17

Homework 2

due February 27, 2017, 2:30pm EST

1 Collision Resistant Hash Functions [4 points]

Let H_0 be a collision resistant hash function defined over $(\mathcal{M}, \{0,1\}^n)$.

(a) **[2 points]** Let H_1 be another collision resistant hash function defined over $(\mathcal{M}, \{0,1\}^n)$ and let $H_2(x) := H_0(x)||H_1(x)$. Is this hash function H_2 more secure than the original hash function H_0 or H_1? Or is it less secure? What about $H_2'(x) := H_0(x) \oplus H_1(x)$? Please argue.

(b) **[2 points]** Use H_0 to construct a hash function H_3 over $(\mathcal{M}, \{0,1\}^n)$ that is also collision resistant, but if one truncates the output of H_3 by one bit then H_3 is no longer collision resistant. That is, H_3 is collision resistant, but $H_4(x) := H_3(x)[0 \ldots n-2]$ is not.

2 Authenticated Encryption [3 points]

Let $SE = (K, E, D)$ be a symmtric encryption scheme and let $MA = (K', MAC, V)$ be a message authentication code. Alice and Bob share a pair of secret keys (k_1, k_2), where $k_1 \in K$ and $k_2 \in K'$. Alice wants to send a message m to Bob in such a way that the message is both secret and authenticated. For each of the following schemes, briefly describe whether the scheme is either secure or not. Also briefly describe whether the scheme can be forgeable or not. You do not need to formally prove your answer, but please justify your answers.

(a) **[0.5 points]** $m, MAC_{k_2}(E_{k_1}(m))$

(b) **[0.5 points]** $E_{k_1}(m, MAC_{k_2}(m))$

(c) **[0.5 points]** $MAC_{k_2}(E_{k_1}(m))$

(d) **[0.5 points]** $E_{k_1}(m), MAC_{k_2}(m)$

(e) **[0.5 points]** $E_{k_1}(m), E_{k_1}(MAC_{k_2}(m))$

(f) **[0.5 points]** $E_{k_1}(m), MAC_{k_2}(E_{k_1}(m))$

3 Timing Attacks on MAC Verification [4 points]

Let (S, V) be a deterministic MAC system where tags T are n-bytes long. The verification algorithm $V(k, m, t)$ is implemented as follows: it first computes $t' \leftarrow S(k, m)$ and then does:

```
for $i \leftarrow 0$ to $n-1$ do
    if $t[i] \neq t'[i]$ then
        output reject and exit
    end if
end for
output accept
```

(a) **[2 points]** Show that this implementation is vulnerable to a timing attack. An attacker who can submit arbitrary queries to algorithm V and accurately measure V's response time can forge a valid tag on every message m of its choice with at most $256 \cdot n$ queries to V.

(b) **[2 points]** How would you implement V to prevent the timing attack from part (a)? Please state your verification algorithm V and argue why it is secure against timing attack from part (a).

4 Key Exchange [3 points]

Recall the Diffie-Hellman key exchange protocol from class. The protocol was vulnerable to a Man-In-The-Middle attack however it can be amended if parties A and B post g^a, g^b on some public messageboard where they can be verified, in fact, with this construction the protocol can be run non-interactively.

At this point we might be tempted to feel that this protocol is secure, and in the limited problem setting we have described it does appear to be, but applied cryptography is never that simple. In practice the Diffie-Hellman key exchange protocol is deployed all over the internet, and with such deployment comes some odds and ends that causes the protocol to provide less security than intended.

Read the following paper **Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice** (https://weakdh.org/imperfect-forward-secrecy-ccs15.pdf) and answer the questions below.

(a) **[1 point]** Briefly describe the attack in this paper. In particular, name at least two key problems that allowed the attack to be carried out.

(b) **[2 points]** Develop a solution for the problems described in the previous part and argue for their adpotion. In particular, argue for why your solution is the best solution to this problem in terms of security, usability and ease of adoption.

5 (Programming) Factoring RSA Modulus [6 points]

In RSA, two large primes p, q (say 512 bits each) are selected, and used to form $N = p \cdot q$ called the modulus. The security of RSA requires that given N, finding p, q is intractible for any efficient (poly. time) adversary.

Typically, p and q are generated independently of each other, we will explore what happens if p and q are very close together, and show that if p and q are sufficiently close, then the RSA modulus can be easily factored which breaks the security of RSA. This occasionally happens in practice because the process for generating large prime numbers typically starts with a random seed from which the algorithm scans for primes. After finding the first prime, a new random seed is supposed to be selected, but instead the implementation might simply continue scanning and thus find a second prime located very close to the first.

Let us consider a naive algorithm for factoring N. Start by guessing p to be the smallest odd (since p is prime) 512 bit integer and divide N by the guess for p. If the result is an integer, than the guess for p must have been correct, and the result is q, thus N has been factored, otherwise increment the guess by 2 and try again. This algorithm will take at most $\frac{2^{512}}{2} = 2^{511}$ guesses and on average 2^{510} guesses. This is computationally intractible since p and q were selected sufficiently large.

A more efficient algorithm can be used but requires that we know the mid-point between p and q. Let $A = \frac{p+q}{2}$ be the arithmatic average of the two primes. Since both p and q are odd, $p + q$ is even and A is an integer. We can express $p = A + x$ and $q = A - x$ for some integer x. The new algorithm works by cheking if $(A-x)(A+x) = N$, starting with $x = 1$ and incrementing x on every iteration. We can see that the number of iterations is equal to $\frac{|p-q|}{2}$ or $\frac{|p-q|}{4}$ when skipping even numbers, so if p and q are close, this runtime is quite fast.

Suppose you are given a composite $N = pq$ where p and q satisfy:

$$|p - q| < 2N^{1/4}$$

Then it follows that:

$$A - \sqrt{N} < 1$$

Proof:

$$A^2 - N = \left(\frac{p+q}{2}\right)^2 - N = \frac{p^2+2N+q^2}{4} - N = \frac{p^2-2N+q^2}{4} = \frac{(p-q)^2}{4}$$

useful fact: $\forall x, y : (x-y)(x+y) = x^2 - y^2$

$$A - \sqrt{N} = (A - \sqrt{N})\frac{A+\sqrt{N}}{A+\sqrt{N}} = \frac{A^2 - N}{A+\sqrt{N}} = \frac{\frac{(p-q)^2}{4}}{A+\sqrt{N}}$$

fact: $\sqrt{N} \leq A$

$$A - \sqrt{N} \leq \frac{\frac{(p-q)^2}{4}}{2\sqrt{N}} = \frac{(p-q)^2}{8\sqrt{N}}$$

We assumed that $|p-q| < 2N^{1/4}$ thus $(p-q)^2 < 4\sqrt{N}$, thus:

$$A - \sqrt{N} \leq \frac{4\sqrt{N}}{8\sqrt{N}} = 1/2 \ < 1$$

(a) **[2 points]** The following 2048-bit modulus N is the product of two primes p and q where $|p-q| < 2N^{1/4}$. Factor this modulus and submit your code and its output (the factorization of N). Be careful of rounding and off by 1 errors while scanning, the runtime of this code should be just a few minutes. In `python` the `gmpy2` library can be used for manipulating large numbers, in `C` you can use `GMP`.

$N =$21639472173435216514966277192112338394585770790611623485084657038839759167618076692248319444960304887535087584103301394588416884828859133440554703615822555857180724065447681562217534501105262699290028782912360731532131358071191166807391518837769933803950872347756303383032405482954871817279795412975783004428002167374151137417600372775009121402043733587565888706428232585152571521024975477669300679122065280121955666646369317131318637364970196180083848587680445169995212079868841523900417687042433295290768692729826856191387802957353441516812454802577090996864622314291845208137446949559711931974603613669736338426333

(b) **[2 points]** The following 60-bit modulus N is the product of two primes p and q where $2N^{1/4} < |p-q| < 2^{11}N^{1/4}$ (further apart by a factor of 2^{10}). Factor this modulus and submit your code and its output (the factorization of N).

$$N = 81991444143945857$$

Hint: $A - \sqrt{(N)} < 2^{20}$, so while A does not immediately follow from N, you can scan from $\sqrt{N}$ upwards guessing different values of A. Also note that the modulus size has been reduced signficantly, although this is much more efficient than the naive algorithm, for a 1024 bit modulus, it would still take a single computer quite a while, and the runtime of this algorithm will be several minutes.

(c) **[2 points]** The following 2048 bit modulus N is the product of two primes p and q where $|3p - 2q| < N^{1/4}$. Prove that $\frac{3p+2q}{2}$ is very close to $\sqrt{6N}$. Factor this modulus and submit your proof, the code and its output (the factorization of N). You will have to come up with a new method for factoring N based on this unique constraint. The runtime of your algorithm should be just a few minutes.

$N =$1541336941882116076668159370541880159932602198220326213755163674447239776234464214952
2632788546550746061321097747689154036834223417675607875349837066823700035172759005007
8726541056360454756336669846292696757527477290471212977735239828289485645864778999703
1670002603773951251265615768693075001997006578270384843834554186844964119407620984049
5493680192487464751460402019759167622871111886557245253985942177793208069700396810145
2094414964265143977276924580722096526259512354946983023671971148824964259958876122851
1099492497099699140735372396264074809454084260072571209685527482903424251724310558191
990536981383216868771

6 (Optional) Strong Second-Preimage Resistance [4 points]

Let $H(x, y)$ be a hash function defined over $(X \times Y, T)$ where $X := \{0, 1\}^n$. We say that H is **strong second-preimage resistant**, or simply **strong-SPR**, if no efficient adversary, given a random x in X as input, can output y, x', y' such that $H(x, y) = H(x', y')$ with non-negligible probability.

(a) **[2 points]** Give an example of a strong-SPR hash function, which is not collision resistant.

(b) **[2 points]** Let H be a strong-SPR. Use H to construct a collision resistant hash function H' defined over $(X \times Y, T)$