Secure and Dependable Systems
Jacobs University Bremen
Dr. Florian Rabe

Homework 7
given: 2017-05-10

Homework 7

You have to submit your solutions as announced in the lecture.
Unless mentioned otherwise, all problems are due 2017-05-19, before the lecture.
There will be no deadline extensions unless mentioned otherwise in the lecture.

Problem 7.1 *Asymmetric Encryption* — Points: 4

Implement an asymmetric encryption scheme based on RSA.

It should have the following

- a key generation function that, given $n \in \mathbb{N}$, randomly chooses primes p, q such that $p \cdot q \geq 2^n$, and then picks a random e for which d can be found,
- encryption and decryption functions that use RSA.

Write a unit test that checks the inversion condition: pick an n and an n-bit message, encrypt and decrypt it, and compare the result for equality.

Problem 7.2 *Hash Collisions* — Points: 4

Consider the following (weak) hash function $hash : \{0,1\}^* \to \mathbb{Z}_N$ for $N = 9993201131$: $hash(x)$ is obtained as follows

1. append 0s to x such that its length is a multiple of 32, and split the result into 32-bit blocks $w_1, \ldots, w_n$
2. put $h := 0$
3. [1] for each $i = 1, \ldots, n$, put $h := (h + 2 + w_i)^{1234567} \bmod 9993201131$
4. return h

Using theory and/or brute force, find a collision of $hash$. Show your work (theory and/or program).

Solution: This was a vague problem where multiple different attacks could be tried (like in a real-life attacks).

A good start was to recognize that $hash$ is similar to RSA and use factorization attack. This yields $N = p \cdot q$ with $p = 99961$ and $q = 99971$. We can then obtain $g : x \mapsto x^{454586863} \bmod 9993201131$ as the inverse of $f : x \mapsto x^{1234567} \bmod 9993201131$.

Assume h is the hash of input of m blocks. For arbitrary n with $n \neq m$ and arbitrary $w_1, \ldots, w_{n-1}$, we can try to find a w_n such that $hash(w_1 \ldots w_n) = h$. To do that, we put $h' = hash(w_1 \ldots w_{n-1})$ and solve $h = f(h' + 2 + w_n)$ for w_n, i.e., $w_n = (g(h) - h' - 2) \bmod 9993201131$. Because $2^{33} < 9993201131 < 2^{34}$, there is a small chance that w_n is too big to be a 32-bit number. In that case, we have to try again, e.g., with a different value for w_{n-1}. That eventually yields a collision.

More generally, this construction allows inverting $hash$, which breaks it as a cryptographic hash function.

Problem 7.3 *Password Hashing* — Points: 4

Implement $hash$ from the previous problem as a function that hashes strings by using the ASCII codes of the characters as the values $w_1, \ldots, w_n$.

Assume $hash$ is (foolishly) used to hash passwords without any salting or stretching, and we expect to have access to some hashes in the future. In order to prepare a break-in, build a table for pairs $(hash(s), s)$ for as many strings s as you can so that you can lookup passwords once you have obtained the hashes.

You may work in groups to build larger tables.

Solution: Given that the previous solution allows constructing an inverse to $hash$, it is redundant for an attacker to table it. But the problem makes sense regardless.

To table the function, it is critical to invest into an efficient implementation of $hash$. We should definitely use square-and-multiply for the exponentiation (divide-and-conquer!), and we can even use a precomputed binary representation of 1234567. Moreover, the modulus should be taken after each step, not just once at the end.

[1] The version I showed in the lecture used $h + w_i$ instead of $h + 2 + w_i$. I changed it to protect against attacks involving $w_1 \in \{0, 1\}$.

A number of further optimizations are possible. Most importantly $hash(s_1 \dots s_n)$ can be obtained from $hash(s_1 \dots s_{n-1})$ in one step. So if we have already tabled all hashes for strings of length $n-1$, we can reuse them to table all hashes for strings of length n (dynamic programming!).

To run one iteration of the dynamic program, we can write a function $d(h)$ that takes a hash h for a string s and returns the hashes for the strings sc for all characters c. Because d must be called on a large fixed set of values for h, it parallelizes with essentially no overhead. If run with k CPUs, the parallelized run time decreases by essentially $1/k$.

Problem 7.4 *Bonus Problem*

Points: depending on effort, at most 5% of grade

No deadline for this problem.

Using *hash* from above, find a meaningful English word/sentence that hashes to 0.

I have not checked how difficult this is. If it is very easy, you have to find a very nice long sentence. If it is very difficult, you may also look for pronounceable words that hash to small numbers.

Solution: Combining the ideas from the previous two problems may a good start here.