Security II: Supervision 1
– exercises

Markus Kuhn with edits by David Llewellyn-Jones

Lent 2018 – Part II

Some of the exercises require the implementation of short programs. The model answers use Perl (see Part IB Unix Tools course), but you can use any language you prefer, as long as it supports an arbitrary-length integer type and offers a SHA-1 function. Include both your source code and the required output into your answers.

Before starting any programming exercise, first estimate how many minutes the solution will take you. Please include in your answers both this estimate, as well as the actual time you required.

**Exercise 1:** Briefly explain the use of hash functions in the Bitcoin protocol. For each distinct use, explain the properties required.


**Exercise 2:** Suppose $H : \{0, 1\}^* \rightarrow \{0, 1\}^l(n)$ is a collision resistant hash function.

(a) Find a hash function $H'$ which is preimage resistant, but not second preimage resistant.

(b) Find a hash function $H''$ which is second preimage resistant, but not collision resistant.

**Exercise 3:** Your colleagues urgently need a collision-resistant hash function. Their code contains already an existing implementation of ECBC-MAC, using a block cipher with 256-bit block size. Therefore, they suggest to use ECBC-MAC with fixed keys $K_1 = K_2 = 0^l$ as a hash function. Show that this construction is not even pre-image resistant.

**Exercise 4:** Show how the DES block cipher can be used to build a 64-bit hash function. How difficult is it to find collisions for your construct?

**Exercise 5:** Currently, when you enter a website password, the password is sent to the site over a secure channel, but otherwise in cleartext. Rather than sending your password directly, suppose your browser instead sent the SHA-1 hash of your password concatenated with the website’s base URL. So, if $M$ is your master password, $w$ is the base URL of the site, then the password $p$ sent to the site would be

$$p = \text{SHA1}(M \parallel w).$$

Would this be more or less secure than the existing arrangement? How could the approach be improved?
Exercise 6: A one-time password authentication system generates 6-character passwords formed using only the set of 64 characters ‘a-zA-Z0-9,’. The first of these passwords is hashed with SHA-1, the resulting hash value is truncated to the first 36 bits, which are then used to form the next password.

(a) After approximately how many passwords is there a better than 50% probability that this hash chain has formed a cycle (i.e., passwords start to recur)?

(b) Write a subroutine genpasswd that accepts a password, and then generates a new 6-character password based on the first 36 bits of the SHA-1 hash value of the input password. Chose a programming language that offers a SHA-1 implementation in its standard library.

(c) Write a subroutine that finds two different input passwords that lead to a collision in genpasswd, i.e. in the first 36 bits of SHA-1, and provide an example such a collision. How many passwords did your program have to generate to find a first collision, and in what run-time?

One example collision:

```
$ perl -e 'use Digest::SHA qw(sha1_hex);while(@ARGV)
{print sha1_hex(shift @ARGV),"\n"}' f7KNL4 EBP37l
ee2109291564192a7372f4caa2477af1646bb593
ee2109291ee27e1d3ee028c21cefc5d55312a383
```

(d) Like part (c), but this time your program must operate in a small amount of memory (i.e., the memory it requires must not grow with the number of passwords generated so far). Compare the number of passwords generated and the execution time with part (c).

Exercise 7: Computer Science Tripos Part II, 2016, Paper 8, Question 11 (first half)

(a) Why does the formal security definition for collision-resistant hash functions require a key $s$ and a security parameter $n$, even though most commonly used standard secure hash functions lack such input parameters? [4 marks]

(b) If $h_s : \{0,1\}^* \rightarrow \{0,1\}^{(n)}$ is a collision-resistant hash function, do the following constructions $H_s$ also provide collision-resistant hash functions? Explain your answers. [2 marks each]

(i) $H_s(x) = h_s(x) \parallel x$ (i.e. append $x$)

(ii) $H_s(x) = h_s(x) \parallel \text{LSB}(x)$ (i.e. append least significant bit of $x$)

(iii) $H_s(x) = h_s(x \mid 1)$ (bitwise-or, i.e. set least significant bit of $x$ to 1)

Exercise 8: The following Perl program implements a non-recursive form of the Euclidean algorithm:

```
#!/usr/bin/perl
use bigint; # use arbitrary-length integer type

sub gcd {
    my ($a0, $b0) = @_;
    my ( $a, $b) = (@_); # use arbitrary-length integer type

    while (1) {
```

```
my $q = $a / $b;
if ($a == $b * $q) {
    print "gcd($a0,$b0) = $b\n";
    return $b;
}
($a, $b) =
    ($b, $a-$b*$q);
}

gcd(2250,360);

Modify it, such that it implements a non-recursive form of the extended Euclidean algorithm. To do so, first define two additional local variables

my ($aa, $ab) = (1, 0);
my ($ba, $bb) = (0, 1);

that record how $a$ and $b$ can be represented as linear combinations of their initial values $a0$ and $b0$, by maintaining the following invariant:

$\begin{align*}
    a & = a0 * aa + b0 * ab \\
    b & = a0 * ba + b0 * bb
\end{align*}$

(a) Extend the final 2-tuple assignment ($a$, $b$) = ($b$, $a-$b*$q$); into a 6-tuple assignment ($a$, $a$aa, $a$ab, $b$, $b$ba, $b$bb) = ($b$, ... ); that maintains the above invariant.

(b) Extend the print and return statements to output the gcd result also as a linear combination of the input values.

(c) If your function is called with egcd(2250,360) it should output

$$gcd(2250,360) = 90 = 2250 * 1 + 360 * -6$$

What is the output of your function if called with the following values?

$$gcd(73381016255931844845,1187329547587210582322)$$

**Exercise 9:** Show how the following two basic properties of every group $(G,\bullet)$ follow from the group axioms given on slide 55:

(a) The neutral element of any group is unique. In other words: if both $e$ and $e'$ are neutral elements of the group, with $g \bullet e = g = e \bullet g$ and $g \bullet e' = g = e' \bullet g$ for every group element $g$, then show that this implies $e = e'$.

(b) The inverse element of any group element is unique. In other words: if $e$ is the neutral element of a group and if we have group elements $f, h$ where $f$ and $h$ are inverse elements of $g$, that is $g \bullet f = e = f \bullet g$ and $g \bullet h = e = h \bullet g$, show that this implies $f = h$.

**Exercise 10:**

(a) Convert your implementation of the extended Euclidean algorithm from Exercise 6 into an implementation of a function modinv(a, n) that returns $a^{-1}$ such that $aa^{-1} \mod n = 1$, or aborts with an error if no such $a^{-1}$ exists. Verify that it outputs

$$modinv(806515533049393, 1304969544928657) = 806515533049393$$

and fails for

$$modinv(4505490,7290036)$$
(b) Which calculation steps of the extended Euclidean algorithm can be dropped for this application?

(c) What is modinv(892302390667940581330701, 1208925819614629174706111)?

**Exercise 11:** Implement a function `modexp(g, e, m)` that calculates $g^e \mod m$ using the square-and-multiply algorithm for modular exponentiation. Test your implementation on

$$123456789^{987654321} \mod (2^{80} - 1) = 785446763117418429158664$$

and then use it to calculate

$$(7^{2^{521} - 1} \mod (2^{3217} - 1)) \mod 10^8$$