Security II: Cryptography – exercises

Markus Kuhn

Lent 2015 – 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 of 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: Explain the collision resistance requirement for the hash function used in a digital signature scheme.

Exercise 2: 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 $K_1 = K_2 = 0^\ell$ as a hash function. Show that this construction is not even pre-image resistant.

Exercise 3: Show how the DES block cipher can be used to build a 64-bit hash function. Is the result collision resistant?

Exercise 4: 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 5 bytes, which are then used to form the next password.

- After how many passwords is there a better than 50% probability that this hash chain has entered a cycle?
- Write a program that finds two passwords that lead to a collision in the first 5 bytes of SHA-1 and provide an example collision. Chose a programming language that offers a SHA-1 implementation in its standard library. One example collision:

```
$ perl -e 'use Digest::SHA qw(sha1_hex);while($_=shift @ARGV)
{print sha1_hex($_),"\n"}' scqfNI KAHgNI
c379d58af1012b03a5b72f5430b663c730499508
c379d58af1123a44e7a1828eaa42a995e44a4634
```

Exercise 5: Use Euclid's algorithm to calculate $\gcd(36, 24)$.

Exercise 6: 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) = @_;

    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:

```
$a == $a0 * $aa + $b0 * $ab
$b == $a0 * $ba + $b0 * $bb
```

(a) Extend the final 2-tuple assignment `($a, $b) = ($b, $a-$b*$q);` into a 6-tuple assignment `($a, $aa, $ab, $b, $ba, $bb) = ($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(733810016255931844845,1187329547587210582322)
```

Exercise 7: Show how the following two basic properties of every group $(\mathbb{G}, \bullet)$ follow from the group axions given on slide 47:

(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 g, f, h where f and h are inverse elements of g, that is $g \bullet f = e = f \bullet g$ and $g \bullet h = e = g \bullet h$, show that this implies $f = h$.

Exercise 8: Let $(\mathbb{F}, \boxplus, \boxtimes)$ be a field. The definition of a field requires that $\boxtimes$ is left-distributive over $\boxplus$, which means that for any $a, b, c \in \mathbb{F}$: $a \boxtimes (b \boxplus c) = (a \boxtimes b) \boxplus (a \boxtimes c)$. Show that this requirement implies the right-distributive property $(a \boxplus b) \boxtimes c = (a \boxtimes c) \boxplus (b \boxtimes c)$.

Exercise 9:

(*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 $aa^{-1} \bmod n = 1$, 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 10: Use Euler's theorem to calculate the inverse

(*a*) $5^{-1} \bmod 7$

(*b*) $5^{-1} \bmod 12$

(*c*) $5^{-1} \bmod 15$

Exercise 11: Given an abelian group $(\mathbb{G}, \bullet)$, let $\mathbb{H}$ be the set of its quadratic residues, that is $\mathbb{H} = \{g^2 \mid g \in \mathbb{G}\}$. Show that $(\mathbb{H}, \bullet)$ is a subgroup of $(\mathbb{G}, \bullet)$.

Exercise 12: Name two advantages of using cyclic groups of *prime order* in cryptographic schemes that rely on the difficulty of the Discrete Logarithm problem or the Diffie–Hellman problems.

Exercise 13: Slide 66 shows how to construct a prime-order subgroup $\mathbb{G} \subset \mathbb{Z}_p^*$, for use with cryptosystems that rely on the discrete-logarithm problem being hard. It contains half of all elements of $\mathbb{Z}_p^*$, namely the $|\mathbb{G}| = q = (p-1)/2$ quadratic residues (with p, q prime). However, sometimes we want to choose q much smaller than p.

(a) What different criteria apply to choosing the bit-length of p and q, and what are the main advantages of having q much smaller than p?

(b) We can construct a smaller subgroup $\mathbb{G} \subset \mathbb{Z}_p^*$ with prime order $q = (p-1)/r$ for any pair of primes p, q with $p = rq + 1$ (Schnorr group). For arbitrary $h \in \mathbb{Z}_p^* \setminus \{1\}$, the value $g = h^r \bmod p$ will be a generator of this group. (The construction on slide 66 merely shows the case $r = 2$).

(i) Show that $\mathbb{G} = \{h^r \bmod p | h \in \mathbb{Z}_p^*\}$ is a subgroup of $\mathbb{Z}_p^*$.

(ii) Show that $\mathbb{G} = \{h^r \bmod p | h \in \mathbb{Z}_p^*\}$ has $q = (p-1)/r$ elements, by showing that the function $f_r : \mathbb{Z}_p^* \to \mathbb{G}$ with $f_r(x) = x^r \bmod p$ is an r-to-1 function.

[*Hint:* Let g be a generator of $\mathbb{Z}_p^*$ such that $\{g^0, g^1, \ldots, g^{p-2}\} = \mathbb{Z}_p^*$. Under what condition for i, j is $(g^i)^r \equiv (g^j)^r \pmod{p}$? For any fixed $j \in \{0, \ldots, p-2\} = \mathbb{Z}_{p-1}$, what values of $i \in \mathbb{Z}_{p-1}$ fulfill that condition, and how many such values i are there?]

(iii) Sometimes, if we receive a value a from an untrusted source, we should first verify that $a \in \mathbb{G} = \{h^r \bmod p | h \in \mathbb{Z}_p^*\}$ before using it further. Show that for any $a \in \mathbb{Z}_p^*$ we have $a \in \mathbb{G}$ if and only if $a^q \bmod p = 1$.

[*Hint:* Assume $a = g^i$ where g is a generator of $\mathbb{Z}_p^*$ and $i \in \{0, \ldots, p-2\}$. Then show that $a^q \equiv 1 \bmod p$ iff $r|i$.]

Exercise 14: Implement a function `modexp(g, e, m)` that calculates $g^e \bmod m$ using the square-and-multiply algorithm for modular exponentiation. Test your implementation on

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

and then use it to calculate

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

Exercise 15: Let $\mathcal{G}(1^\ell)$ be a polynomial-time group generator that outputs an ℓ-bit prime p and a generator g of $\mathbb{Z}_p^*$. Show that the DDH problem is not hard relative to $\mathcal{G}$.
[*Hint:* Recall that Euler's criterion allows efficient detection of quadratic residues.]

Exercise 16: With RSA encryption, it is common practice to choose e as a small number (e.g., 3, 17, $2^{16} + 1$).

(a) How does this affect the speed of encryption?

(b) If you wanted to make decryption faster, could you simply set d to one of these three values instead?

(c) How else can RSA decryption be calculated more efficiently using the Chinese Remainder Theorem and Fermat's little theorem?

Exercise 17: In the textbook RSA encryption scheme, with $n = pq$ being a product of two different primes and $ed \bmod \varphi(n) = 1$, the identity $m^{ed} \bmod n = m$, which states that we obtain the same plaintext m after encryption and decryption, is only guaranteed by Euler's theorem for any $m \in \mathbb{Z}_n^*$, that is if $\gcd(n, m) = 1$.

(a) Show that it actually also holds for any $m \in \mathbb{Z}_n$. [Hint: CRT]

(b) Conversely, if we instead had chosen $n = p^2$ being the square of a prime number (i.e., $p = q$), show a simple example for the fact that in this case $ed \bmod \varphi(n) = 1$ does *not* imply $m^{ed} \bmod n = m$ for all $m \in \mathbb{Z}_n$.

Exercise 18: A device vendor uses the DSA signature scheme to digitally sign configuration updates. The system parameters are

$$\begin{aligned} p &= \texttt{0x8df2a494492276aa3d25759bb06869cbeac0d83afb8d0cf7cbb8324f0d7882e5} \\ &\quad\ \texttt{d0762fc5b7210eafc2e9adac32ab7aac49693dfbf83724c2ec0736ee31c80291} \\ q &= \texttt{0xc773218c737ec8ee993b4f2ded30f48edace915f} \\ g &= \texttt{0x626d027839ea0a13413163a55b4cb500299d5522956cefcb3bff10f399ce2c2e} \\ &\quad\ \texttt{71cb9de5fa24babf58e5b79521925c9cc42e9f6f464b088cc572af53e6d78802} \end{aligned}$$

and the vendor's public key is

$$\begin{aligned} y &= \texttt{0xeb772a91db3b69af90c5da844d7733f24270bdd11aac373b26f58ff528ef2678} \\ &\quad\ \texttt{94b1e746e3f20b8b89ce9e5d641abbff3e3fa7dedd3264b1b313d7cd569656c} \end{aligned}$$

The vendor has already signed two messages:

$$\begin{aligned} H(m_1) &= \text{SHA-1}(\texttt{"Monday"}) = \texttt{0x932eeb1076c85e522f02e15441fa371e3fd000ac} \\ r_1 &= \texttt{0x8f4378d1b2877d8aa7c0687200640d4bba72f2e5} \\ s_1 &= \texttt{0x696de4ffb102249aef907f348fb10ca704a4b186} \\ H(m_2) &= \text{SHA-1}(\texttt{"Tuesday"}) = \texttt{0x42e43b612a5dfae57ddf5929f0fb945ae83cbf61} \\ r_2 &= \texttt{0x8f4378d1b2877d8aa7c0687200640d4bba72f2e5} \\ s_2 &= \texttt{0x25f87cbb380eb4d7244963e65b76677bc968297e} \end{aligned}$$

(a) Calculate $g^q \bmod p$.

(b) Verify that the two signatures are valid under the given public key y. (Preferably perform the required calculations using the `modinv` and `modexp` routines that you implemented yourself in exercises 9 and 14. Alternatively, download a computer-algebra system, such as Sage or PARI/GP.)

(c) What mistake did the vendor make when generating these two signatures?

(d) Exploit this mistake to reconstruct the secrets k and x used to generate these signatures. [*Hint:* Start by subtracting the two defining equations for s_1 and s_2 from each other.]

(e) Use this information to falsify a signature for the new message

$$H(m_3) = \text{SHA-1}(\texttt{"Wednesday"}) = \texttt{0x932eeb1076c85e522f02e15441fa371e3fd000ac}$$

and then verify its correctness against public key y.