Cryptography
– exercises

Markus Kuhn

Lent 2020 – CST 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.

1 Historic ciphers

Exercise 1: Decipher the shift cipher text
LUXDZNUAMNDODJUDTUZDGYQDLUXDGOJDCKDTKKJDOZ

Exercise 2: How can you break any transposition cipher with $\lceil \log_a n \rceil$ chosen plaintexts, if a is the size of the alphabet and n is the permutation block length?

2 Perfect secrecy

Exercise 3: Show that the shift cipher provides unconditional security if $\forall K \in \mathbb{Z}_{26}$: $\mathbb{P}(K) = 26^{-1}$ for plaintexts $M \in \mathbb{Z}_{26}$.

Exercise 4: Show that an encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ over a message space $\mathcal{M}$ is *perfectly secret* if and only if

(*a*) for every probability distribution over $\mathcal{M}$, every message $M \in \mathcal{M}$, and every ciphertext $C \in \mathcal{C}$ with $\mathbb{P}(C) > 0$ we have

$$\mathbb{P}(C|M) = \mathbb{P}(C).$$

(*b*) for every probability distribution over $\mathcal{M}$, every message pair $M_0, M_1 \in \mathcal{M}$, and every ciphertext $C \in \mathcal{C}$ with $\mathbb{P}(C) > 0$ we have

$$\mathbb{P}(C|M_0) = \mathbb{P}(C|M_1).$$

3 Semantic security

4 Block ciphers

Exercise 5: If the round function f in a Feistel construction is a pseudo-random function, how many rounds r are at least necessary to build a pseudo-random permutation? What test can you apply to distinguish a Feistel structure with $r - 1$ rounds (with high probability) from a random permutation?

Exercise 6: Using a given pseudo-random function $F : \{0,1\}^{100} \to \{0,1\}^{100}$, construct a pseudo-random permutation $P : \{0,1\}^{300} \to \{0,1\}^{300}$ by extending the Feistel principle appropriately.

Exercise 7: What happens to the ciphertext block if all bits in both the key and plaintext block of DES are inverted?

Exercise 8: Given a hardware implementation of the DES encryption function, what has to be modified to make it decrypt?

5 Modes of operation

Exercise 9: In the CBC mode of operation, the initial vector (IV) is chosen uniformly at random, using a secure source of random bits. Show that CBC would not be CPA secure if the initial vector could be anticipated by the adversary, for example because it is generated instead using a counter or a time-stamp.

Exercise 10: Explain for each of the discussed modes of operation (ECB, CBC, CFB, OFB, CTR) of a block cipher how decryption works.

Exercise 11: A sequence of plaintext blocks $M_1, \ldots, M_8$ is encrypted using DES into a sequence of ciphertext blocks. Where an IV is used, it is numbered C_0. A transmission error occurs and one bit in ciphertext block C_3 changes its value. As a consequence, the receiver obtains after decryption a corrupted plaintext block sequence $M'_1, \ldots, M'_8$. For the discussed modes of operation (ECB, CBC, CFB, OFB, CTR), how many bits do you expect to be wrong in each block M'_i?
(Hint: You may find it helpful to draw decryption block diagrams.)

Exercise 12: Your opponent has invented a new stream-cipher mode of operation for 128-bit key AES. He thinks that OFB could be improved by feeding back into the key port rather than the data port of the AES chip. He therefore sets $R_0 = K$ and generates the key stream by $R_{i+1} = E_{R_i}(R_0)$. Is this better or worse than OFB?

Exercise 13: A programmer wants to use CBC in order to protect both the integrity and confidentiality of network packets. She attaches a block of zero bits M_{n+1} to the end of the plaintext $M_1 \| \ldots \| M_n$ as redundancy, then encrypts with CBC. At the receiving end, she verifies that the added redundant bits are still all zero after CBC decryption. Does this test ensure the integrity of the transferred message?

6 Message authenticity

Exercise 14:

Show that CTR mode is not CCA secure.

7 Authenticated encryption

Exercise 15: Your colleagues have invented a new authenticated encryption scheme that they call AES-CBC+CMAC. Their key generating function outputs a 128-bit AES key K, and their encryption function outputs $C\|T = \mathsf{Enc}_K(M)\|\mathsf{Mac}_K(M)$, where $\mathsf{Enc}_K(M)$ shall be the AES-CBC encryption of M with key K (with random IV each time), and $\mathsf{Mac}_K(M)$ shall be the AES-CMAC of M with key K. Show that this construct lacks CPA security.

8 Secure hash functions

Exercise 16: Explain the purpose of the collision-resistance requirement for the hash function used in a digital signature scheme.

Exercise 17: 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 18: 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?

9 Secure hash applications

Exercise 19: 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*).

10 Key distribution problem

11 Number theory and group theory

Exercise 20: Use Euclid's algorithm to calculate gcd(36, 24).

Exercise 21: 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 22: Show how the following two basic properties of every group $(\mathbb{G}, \bullet)$ follow from the group axioms given on slide 158:

(*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 $g \bullet h = e = h \bullet g$, show that this implies $f = h$.

Exercise 23: 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 24:

(*a*) Convert your implementation of the extended Euclidean algorithm from Exercise 21 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 25: Use Euler's theorem to calculate the inverse

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

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

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

Exercise 26: 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 27: 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

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

12 Discrete logarithm problem

Exercise 28: 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 29: Which elliptic curve is used for the digital signatures used to sign transactions in the Bitcoin blockchain and what are some of its properties?

13 RSA trapdoor permutation

Exercise 30: 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 31: 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 $m \in \mathbb{Z}_n$.

14 Digital signatures

<u>Exercise 32:</u> 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 24 and 27. 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

$$H(m_3) = \text{SHA-1}(\texttt{"Wednesday"}) = \texttt{0x5656b9b79b0316fc611a9c30d2ffac25228b8371}$$

and then verify its correctness against public key y.