Christof Paar · Jan Pelzl · Tim Güneysu

Understanding Cryptography

Solution Handbook

(odd numbered Problems)

Second Edition

Problems of Chapter 1

1.1

1. Letter frequency analysis of the ciphertext:

letter	count	freq [%]	letter	count	freq [%]
A	5	0.77	N	17	2.63
B	68	10.53	O	7	1.08
C	5	0.77	P	30	4.64
D	23	3.56	Q	7	1.08
E	5	0.77	R	84	13.00
F	1	0.15	S	17	2.63
G	1	0.15	T	13	2.01
H	23	3.56	U	24	3.72
I	41	6.35	V	22	3.41
J	48	7.43	W	47	7.28
K	49	7.59	X	20	3.10
L	8	1.24	Y	19	2.94
M	62	9.60	Z	0	0.00

2. `Because the practice of the basic movements of kata is the focus and mastery of self is the essence of Matsubayashi Ryu karate do, I shall try to elucidate the movements of the kata according to my interpretation based on forty years of study.`

   `It is not an easy task to explain each movement and its significance, and some must remain unexplained. To give a complete explanation, one would have to be qualified and inspired to such an extent that he could reach the state of enlightened mind capable of recognizing soundless sound and shapeless shape. I do not deem myself the final authority, but my experience with kata has left no doubt that the following is the proper application and interpretation. I offer my theories in the hope that the essence of Okinawan karate will remain intact.`
3. Shoshin Nagamine, further reading: *The Essence of Okinawan Karate-Do* by Shoshin Nagamine, Tuttle Publishing, 1998.

1.3

One search engine costs $ 100 including overhead. Thus, 1 million dollars buy us 10,000 engines.

1. key tests per second: $5 \cdot 10^8 \cdot 10^4 = 5 \cdot 10^{12}$ keys/sec
   On average, we have to check $(2^{127}$ keys:
   $(2^{127}\text{keys})/(5 \cdot 10^{12}\text{keys/sec}) = 3.40 \cdot 10^{25}\text{sec} = 1.08 \cdot 10^{18}\text{years}$

That is about $10^8 = 100,000,000$ times longer than the age of the universe. Good luck.

2. Let i be the number of Moore iterations needed to bring the search time down to 24h:
   $1.08 \cdot 10^{18}\text{years} \cdot 365/2^i = 1\text{day}$
   $2^i = 1,08 \cdot 10^{18} \cdot 365\text{days}/1\text{day}$
   $i = 68.42$
   We round this number up to 69 assuming the number of Moore iterations is discreet. Thus, we have to wait for:
   $1.5 \cdot 69 = 103.5$ years
   Note that it is extremely unlikely that Moore's Law will be valid for such a time period! Thus, a 128-bit key seems impossible to brute-force, even in the foreseeable future.

1.5

1. On the last field there are 2^{63} grains
2. The amount is $\frac{\frac{2^{63} \cdot 0,03}{1000000}}{480000000} = 576.46$ times the yearly rice grain yield.
3. $2^{10} \cdot 0,1mm = 102,4mm = 10,24cm$
4. $1km = 10^6 mm \Rightarrow log_2(10^7) = 23,25 \Rightarrow$ The paper needs to be folded 24 times.
5. $log_2(10^7 \cdot 384400) = 41,81 \Rightarrow$ The paper needs to be folded at least 42 times.
6. $log_2(10^7 \cdot 9460730472580.8) = 66,36 \Rightarrow$ The paper needs to be folded at least 67 times.

1.7

1. $15 \cdot 29 \bmod 13 \equiv 2 \cdot 3 \bmod 13 \equiv 6 \bmod 13$
2. $2 \cdot 29 \bmod 13 \equiv 2 \cdot 3 \bmod 13 \equiv 6 \bmod 13$
3. $2 \cdot 3 \bmod 13 \equiv 2 \cdot 3 \bmod 13 \equiv 6 \bmod 13$
4. $2 \cdot 3 \bmod 13 \equiv 2 \cdot 3 \bmod 13 \equiv 6 \bmod 13$

15, 2 and -11 (and 29 and 3 respectively) are representations of the same equivalence class modulo 13 and can be used "synonymously".

1.9

1.

Multiplication table for Z_4

×	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	0	2
3	0	3	2	1

2.

Addition table for Z_5

+	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

Multiplication table for Z_5

×	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

3.

Addition table for Z_6

+	0	1	2	3	4	5
0	0	1	2	3	4	5
1	1	2	3	4	5	0
2	2	3	4	5	0	1
3	3	4	5	0	1	2
4	4	5	0	1	2	3
5	5	0	1	2	3	4

Multiplication table for Z_6

×	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	2	3	4	5
2	0	2	4	0	2	4
3	0	3	0	3	0	3
4	0	4	2	0	4	2
5	0	5	4	3	2	1

4. Elements without a multiplicative inverse in Z_4 are 2 and 0
   Elements without a multiplicative inverse in Z_6 are 2, 3, 4 and 0
   For all nonzero elements of Z_5 exists because 5 is a prime. Hence, all nonzero elements smaller than 5 are relatively prime to 5.

1.11

1. $x = 9 \bmod 13$
2. $x = 7^2 = 49 \equiv 10 \bmod 13$
3. $x = 3^{10} = 9^5 \equiv 81^2 \cdot 9 \equiv 3^2 \cdot 9 \equiv 81 \equiv 3 \bmod 13$
4. $x = 7^{100} = 49^{50} \equiv 10^50 \equiv (-3)^{50} = (3^{10})^5 \equiv 3^5 \equiv 3^2 = 9 \bmod 13$
5. by trial: $7^5 \equiv 11 \bmod 13$

1.13

1. FIRST THE SENTENCE AND THEN THE EVIDENCE SAID THE QUEEN
2. Charles Lutwidge Dodgson, better known by his pen name Lewis Carroll

1.15

$$a \equiv (x_1 - x_2)^{-1}(y_1 - y_2) \bmod m$$
$$b \equiv y_1 - a x_1 \bmod m$$

The inverse of $(x_1 - x_2)$ must exist modulo m, i.e., $\gcd((x_1 - x_2), m) = 1$.

1.17

1. $\{9,0,12,0,8,10,0\}$
2. LOPEJBEJKQRA
   Example S: $18 + 9 \bmod 26 = 1$ = B
3. One can consider the Vigenère cipher as consisting of l shift ciphers that are used cyclically and it is, thus, barely more secure than the standard shift cipher. Here is an attack:

   Let's assume the adversary knows the value of l. He then sorts the ciphertext in l bins: the first containing the letters $c_0, c_l, c_{2l}, \ldots$, the second bin contains $c_1, c_{l+1}, c_{2l+1}, \ldots$, and so on. Since each bin consists of letters that have been shifted by the same key value k_i, we can now apply normal frequency analysis to each bin and will easily find all key values $k_0, k_1, \ldots, k_{l-1}$.
   The remaining problem is how the adversary obtains l. In practice, the easiest approach is to simply guess different values for l. In the first iteration, the attacker assumes $l = 1$ and performs the attack and checks whether he obtains valid plaintext. If not, he assumes $l = 2$, performs the attack and again checks for a valid plaintext. Most likely, incorrect values for l will not give meaningful plaintexts but the correct l value will immediately be recognizable by a correct (and meaningful) plaintext.

Problems of Chapter 2

2.1

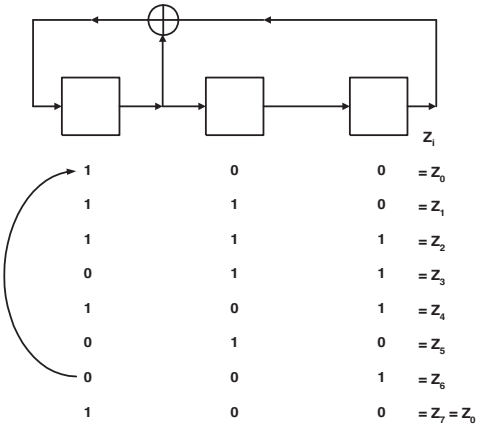
1. $y_i = x_i + K_i \bmod 26$
   $x_i = y_i - K_i \bmod 26$
   The keystream is a sequence of random integers from Z_{26}.
2. $x_1 = y_1 - K_1 =$ "B" $-$ "R" $= 1 - 17 = -16 \equiv 10 \bmod 26 =$ "K" etc $\cdots$
   Decrypted Text: "KASPAR HAUSER"
3. He was knifed.

2.3 We need 128 pairs of plaintext and ciphertext *bits* (i.e., 16 byte) in order to determine the key. s_i is being computed by
$s_i = x_i \bigoplus y_i;\ \ i = 1,2,\cdots,128.$

2.5 Plaintext:
`4c 65 74 73 45 6e 63 72 79 70 74 54 68 69 73 42 6f 6f 6b`
= LetsEncryptThisBook

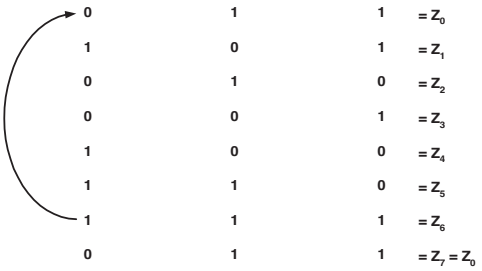
2.7

1.



Sequence 1: z_0 = 0 0 1 1 1 0 1 0 0 1 1 1 0 1 ...

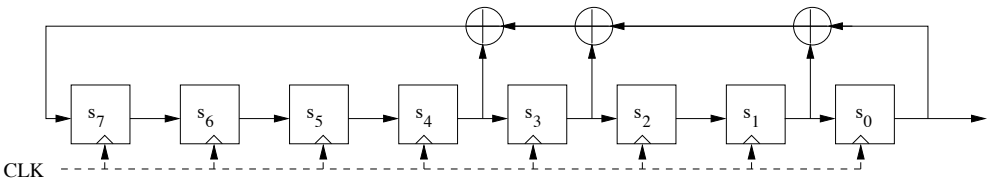
2.



Sequence 2: z_0 = 1 1 0 1 0 0 1 1 1 0 1 0 0 1 ...

3. The two sequences are shifted versions of one another.

2.9 The feedback polynomial from 2.2 is $x^8+x^4+x^3+x^2+1$. So, the resulting first



1	1	1	1	1	1	1	1	$=Z_0$
0	1	1	1	1	1	1	1	$=Z_1$
0	0	1	1	1	1	1	1	$=Z_2$
0	0	0	1	1	1	1	1	
0	0	0	0	1	1	1	1	
1	0	0	0	0	1	1	1	
0	1	0	0	0	0	1	1	
0	0	1	0	0	0	0	1	
1	0	0	1	0	0	0	0	
1	1	0	0	1	0	0	0	
1	1	1	0	0	1	0	0	
0	1	1	1	0	0	1	0	
0	0	1	1	1	0	0	1	
1	0	0	1	1	1	0	0	
0	1	0	0	1	1	1	0	$=Z_{14}$
0	0	1	0	0	1	1	1	$=Z_{15}$

two output bytes are $(1001000011111111)_2 = (90FF)_{16}$.

2.11

1. The attacker needs 512 consecutive plaintext/ciphertext bit pairs x_i, y_i to launch a successful attack.
2. a. First, the attacker has to monitor the previously mentioned 512-bit pairs.
   b. The attacker calculates $s_i = x_i + y_i \bmod 2$, $i = 0, 1, \ldots, 2m-1$
   c. In order to calculate the (secret) feedback coefficients p_i, Oscar generates 256 linearly dependent equations using the relationship between the unknown key bits p_i and the keystream output defined by the equation

$$s_{i+m} \equiv \sum_{j=0}^{m-1} p_j \cdot s_{i+j} \bmod 2; s_i, p_j \in \{0,1\}; i = 0,1,2,...,255$$

   with $m = 256$.
   d. After generating this linear equation system, it can be solved e.g., using Gaussian Elimination, revealing the 256 feedback coefficients.

3. The key of this system is represented by the 256 feedback coefficients. Since the initial contents of the LFSR are unalteredly shifted out of the LFSR and XORed with the first 256 plaintext bits, it would be easy to calculate them.

2.13
$x_i \bigoplus y_i = x_i \bigoplus (x_i \bigoplus z_i) = z_i$

$W \Longleftrightarrow 22 = 10110_2$	$J \Leftrightarrow 9 = 01001_2$
$P \Longleftrightarrow 15 = 01111_2$	$5 \Longleftrightarrow 31 = 11111_2$
$I \Leftrightarrow 8 = 01000_2$	$A \Leftrightarrow 0 = 00000_2$

$$\begin{array}{l} x_i = 10110\,01111\,01000 \\ \underline{y_i = 01001\,11111\,00000} \\ z_i = 11111\,10000\,01000 \end{array}$$

1. Initialization Vector: $(Z_0 = 1,1,1,1,1,1)$
2.

$$\begin{pmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \\ C_4 \\ C_5 \end{pmatrix} = \begin{pmatrix} 1&1&1&1&1&1 \\ 1&1&1&1&1&0 \\ 1&1&1&1&0&0 \\ 1&1&1&0&0&0 \\ 1&1&0&0&0&0 \\ 1&0&0&0&0&0 \end{pmatrix}^{-1} \cdot \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

$$= \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

3.

	$\overbrace{01001}^{J}$	$\overbrace{11111}^{5}$	$\overbrace{00000}^{A}$	$\overbrace{11010}^{0}$	$\overbrace{00100}^{E}$	$\overbrace{00011}^{D}$	$\overbrace{01001}^{J}$	$\overbrace{11100}^{2}$	$\overbrace{00001}^{B}$
$y_i =$	01001	11111	00000	11010	00100	00011	01001	11100	00001
$z_i =$	11111	10000	01000	01100	01010	01111	01000	11100	10010
$x_i =$	$\underbrace{10110}_{W}$	$\underbrace{01111}_{P}$	$\underbrace{01000}_{I}$	$\underbrace{10110}_{W}$	$\underbrace{01110}_{O}$	$\underbrace{01100}_{M}$	$\underbrace{00001}_{B}$	$\underbrace{00000}_{A}$	$\underbrace{10011}_{T}$

4. Wombats live in Australia.
5. Known-plaintext attack.

2.15 The keystream is given by $z_i = 4, z_{i+1} = 228, z_{i+2} = 187, z_{i+3} = 21, z_{i+4} = 148$ and yields the following system of equations:

$$
\begin{aligned}
z_{i+2} &\equiv z_{i+1} \cdot a + z_i \cdot b + c \bmod m \quad (I) \\
z_{i+3} &\equiv z_{i+2} \cdot a + z_{i+1} \cdot b + c \bmod m \quad (II) \\
z_{i+4} &\equiv z_{i+3} \cdot a + z_{i+2} \cdot b + c \bmod m \quad (III) \\
&\quad \text{inserting the values yields a system of linear eqn.:} \\
187 &\equiv 228 \cdot a + 4 \cdot b + c \bmod m \quad (I) \\
21 &\equiv 187 \cdot a + 228 \cdot b + c \bmod m \quad (II) \\
148 &\equiv 21 \cdot a + 187 \cdot b + c \bmod m \quad (III) \\
&\quad \text{solving the system results in:} \\
\Rightarrow a &\equiv 132 \bmod m \\
\Rightarrow b &\equiv 246 \bmod m \\
\Rightarrow c &\equiv 204 \bmod m
\end{aligned}
$$

2.17 The output of QR is computed as follows:

$$
\begin{aligned}
a &= a + b \\
&= 0x00000001 \\
d &= ROTL^{16}(d \oplus a) \\
&= ROTL^{16}(0x00000001) \\
&= 0x00010000 \\
c &= c + d \\
&= 0x00010000 \\
b &= ROTL^{12}(b \oplus c) \\
&= ROTL^{12}(0x00010000) \\
&= 0x10000000 \\
a &= a + b \\
&= 0x00000001 + 0x10000000 \\
&= 0x10000001 \\
d &= ROTL^{8}(d \oplus a) \\
&= ROTL^{8}(0x00010000 \oplus 0x10000001) \\
&= ROTL^{8}(0x10010001) \\
&= 0x01000110 \\
c &= c + d \\
&= 0x00010000 + 0x01000110 \\
&= 0x01010110 \\
b &= ROTL^{7}(b \oplus c) \\
&= ROTL^{7}(0x10000000 \oplus 0x01010110) \\
&= ROTL^{7}(0x11010110) \\
&= ROTL^{7}(0x11010110) \\
&= 0x80808808
\end{aligned}
$$

Hence

$$
\begin{aligned}
& QR(0x00000001, 0x00000000, 0x00000000, 0x00000000) \\
= \; & 0x10000001, 0x80808808, 0x01010110, 0x01000110
\end{aligned}
$$

Problems of Chapter 3

3.1

1. $s(x_1) \bigoplus s(x_2) = 1110$
   $s(x_1 \bigoplus x_2) = s(x_2) = 0000 \neq 1110$
2. $s(x_1) \bigoplus s(x_2) = 1001$
   $s(x_1 \bigoplus x_2) = s(x_2) = 1000 \neq 1001$
3. $s(x_1) \bigoplus s(x_2) = 1010$
   $s(x_1 \bigoplus x_2) = s(x_2) = 1101 \neq 1010$

3.3

Let $y = IP(x)$ and $z = IP^{-1}(y)$
With $y = (y_1, y_2, \ldots, y_{64})$ and $z = (z_1, z_2, \ldots, z_{64})$

$$
\begin{array}{lll}
 & x_1 \xrightarrow{IP} y_{40} & \xrightarrow{IP^{-1}} z_1 \\
x_2 & \longrightarrow y_8 & \longrightarrow z_2 \\
x_3 & \longrightarrow y_{48} & \longrightarrow z_3 \\
x_4 & \longrightarrow y_{16} & \longrightarrow z_4 \\
x_5 & \longrightarrow y_{56} & \longrightarrow z_5
\end{array}
$$

3.5

Since (parts of) the key is simply XORed with L_0 in the f-function, the input of the S-boxes and hence, the output of the whole f-function, equals those in the previous problem. $S_1(0) = 14 = 1110$
$S_2(0) = 15 = 1111$
$S_3(0) = 10 = 1010$
$S_4(0) = 7 = 0111$
$S_5(0) = 2 = 0010$
$S_6(0) = 12 = 1100$
$S_7(0) = 4 = 0100$
$S_8(0) = 13 = 1101$

$P(S) = D8D8\ DBBC$
$R_1 = L_0 \oplus P(S) = FFFF\ FFFF \oplus D8D8\ DBBC = 2727\ 2443$
$(L_1, R_1) = FFFF\ FFFF\ 2727\ 2443$ **(1)**

3.7

1. First, the flipped bit is moved to position 8 by the PC-1 permutation: PC-1(1) = 8
   After that, it is shifted shifted one position to the left: LS-1(8) = 7
   The PC-2 permutation chooses bit number 7 for roundkey generation of round one:
   PC-2(7) = 19, thus S-box 4 is affected
   For the next rounds, the results are provided in a short form:

Round 2: $PC-2(LS_1(7)) = PC-2(6) = 10 \longrightarrow$ S-box S2
Round 3: $PC-2(LS_2(6)) = PC-2(4) = 16 \longrightarrow$ S-box S3
Round 4: $PC-2(LS_2(4)) = PC-2(2) = 24 \longrightarrow$ S-box S4
Round 5: $PC-2(LS_2(2)) = PC-2(2) = 28 \longrightarrow$ S-box S8
Round 6: $PC-2(LS_2(28)) = PC-2(26) = 17 \longrightarrow$ S-box S3
Round 7: $PC-2(LS_2(26)) = PC-2(24) = 4 \longrightarrow$ S-box S1
Round 8: $PC-2(LS_2(24)) = PC-2(22) = X \longrightarrow$ no S-box affected
Round 9: $PC-2(LS_1(22)) = PC-2(21) = 11 \longrightarrow$ S-box S2
Round 10: $PC-2(LS_2(21)) = PC-2(19) = 14 \longrightarrow$ S-box S3
Round 11: $PC-2(LS_2(19)) = PC-2(17) = 2 \longrightarrow$ S-box S1
Round 12: $PC-2(LS_2(17)) = PC-2(15) = 9 \longrightarrow$ S-box S2
Round 13: $PC-2(LS_2(15)) = PC-2(13) = 23 \longrightarrow$ S-box S4
Round 14: $PC-2(LS_2(13)) = PC-2(11) = 3 \longrightarrow$ S-box S1
Round 15: $PC-2(LS_2(11)) = PC-2(9) = X \longrightarrow$ no S-box affected
Round 16: $PC-2(LS_1(9)) = PC-2(8) = 18 \longrightarrow$ S-box S3

Note that the input changes to any of the S-boxes quickly spread in subsequent round. For instance, the input change of S2 in Round 2 will affect several S-Boxes in Round 3. This means that the single key bit influences not only the S-boxes listed above but in fact all S-boxes after a few rounds.

2. Due to the fact, that the decryption roundkeys K_{dec}(Round i) are equal to K_{enc}(Round 16-i), the affected S-boxes are also the same, just in reversed order.

3.9

1. $K_{1+i} = K_{16-i}$ for $i = 0,1,...7$.
2. Following (a), two equations are established:

$$\begin{aligned} C_{1+i} &= C_{16-i} \\ D_{1+i} &= D_{16-i} \qquad \text{for } i = 0,1,...,7. \end{aligned}$$

These equations yield

$$\begin{aligned} C_{0,j} &= 0 \text{ und } D_{0,j} = 0 \text{ or} \\ C_{0,j} &= 0 \text{ und } D_{0,j} = 1 \text{ or} \\ C_{0,j} &= 1 \text{ und } D_{0,j} = 0 \text{ oder} \\ C_{0,j} &= 1 \text{ und } D_{0,j} = 1 \qquad \text{for } j = 1,2,...,28. \end{aligned}$$

Hence the four weak keys after PC-1 are given by:

$$\begin{aligned} \hat{K}_{w1} &= [0\ldots0\ \ 0\ldots0] \\ \hat{K}_{w2} &= [0\ldots0\ \ 1\ldots1] \\ \hat{K}_{w3} &= [1\ldots1\ \ 0\ldots0] \\ \hat{K}_{w4} &= [1\ldots1\ \ 1\ldots1] \end{aligned}$$

3. $P(\text{randomly chose a weak key}) = \frac{2^2}{2^{56}} = 2^{-54}$.

3.11
Worst-Case: 2^{56} keys.
Average: $2^{56}/2 = 2^{55}$ keys.

3.13

1. A single DES engine can compute $100 \cdot 10^6$ DES encryptions per second. A COPACOBANA machine can, thus, compute $4 \cdot 6 \cdot 20 \cdot 100 \cdot 10^6 = 4.8 \cdot 10^{10}$ DES encryptions per second. For an average of 2^{55} encryptions for a successfull brute-force attack on DES, $2^{55}/(4.8 \cdot 10^{10}) \approx 750600$ seconds are required (which approximately is 8.7 days).
2. For a successfull average attack in one hour, $8.7\dot{2}4 \approx 209$ machines are required.
3. The machine performs a brute–force attack. However, there might be more powerful analytical attacks which explore weaknesses of the cipher. Hence, the key–search machine provides only an upper security threshold since brute–force is always possible.

3.15

1. The state of PRESENT after the execution of one round is `F000 0000 0000 000F`. Below you can find all intermediate values.

Plaintext	`0000 0000 0000 0000`
Round key	`BBBB 5555 5555 EEEE`
State after KeyAdd	`BBBB 5555 5555 EEEE`
State after S-Layer	`8888 0000 0000 1111`
State after P-Layer	`F000 0000 0000 000F`

2. The round key for the second round is `7FFF F777 6AAA AAAA`. Below you can find all intermediate values.

Key	`BBBB 5555 5555 EEEE FFFF`
Key state after rotation	`DFFF F777 6AAA AAAA BDDD`
Key state after S-box	`7FFF F777 6AAA AAAA BDDD`
Key state after CounterAdd	`7FFF F777 6AAA AAAA 3DDD`
Round key for Round 2	`7FFF F777 6AAA AAAA`

Problems of Chapter 4

4.1

1. The successor of the DES, the AES, was chosen by the NIST by a public proceeding. The purpose of this public contest was to allow broadly evaluation of the candidates by as many research organisations and institutes as possible.
   This strongly contrasts to the development of DES, which was only performed by IBM and the NSA firstly keeping details (e.g., the S-boxes) in secret. DES was published and standardized in 1975.
2. 1/2/97: Call for algorithms, which could potentially lead to the AES. The selection process was governed by the NIST. 8/20/98: 15 algorithms were nominated as candidates for the selection process.
   9.8.1999: 5 algorithms reach the "finals" (Mars, RC6, Rijndael, Serpent, Twofish)
   2.10.2000: NIST elects Rijndael to AES.
3. Rijndael
4. Dr. Vincent Rijmen and Dr. Joam Daemen from Belgium
5. Rijndael supports blocksizes of 128, 192 and 256 bits, as well as key lengths of 128, 192 and 256 bits. In fact, only the version with 128 bits blocksize (and all three key lengths) is called AES.

4.3

Multiplication table for $GF(2^3)$, $P(x) = x^3 + x + 1$

$\times$	0	1	x	$x+1$	x^2	x^2+1	x^2+x	x^2+x+1
0	0	0	0	0	0	0	0	0
1	0	1	x	$x+1$	x^2	x^2+1	x^2+x	x^2+x+1
x	0	x	x^2	x^2+x	$x+1$	1	x^2+x+1	x^2+1
$x+1$	0	$x+1$	x^2+x	x^2+1	x^2+x+1	x^2	1	x
x^2	0	x^2	$x+1$	x^2+x+1	x^2+x	x	x^2+1	1
x^2+1	0	x^2+1	1	x^2	x	x^2+x+1	$x+1$	x^2+x
x^2+x	0	x^2+x	x^2+x+1	1	x^2+1	$x+1$	x	x^2
x^2+x+1	0	x^2+x+1	x^2+1	x	1	x^2+x	x^2	$x+1$

4.5 Multiplication in GF(2^4):

1. $A(x) * B(x) = (x^2+1)(x^3+x^2+1) = x^5+x^4+x^2+x^3+x^2+1$
   $A(x) * B(x) = x^5+x^4+x^3+1$

$$
\begin{array}{rrrrrrrrr}
 & & & & & & & x & +1 \\
x^4+x+1 & x^5 & +x^4 & +x^3 & & & & & +1 \\
 & x^5 & & & & +x^2 & +x & & \\
 & & x^4 & +x^3 & +x^2 & +x & +1 & & \\
 & & x^4 & & & & +x & +1 & \\
 & & & x^3 & +x^2 & & & &
\end{array}
$$

$C = x^3 + x^2 \equiv A(x) * B(x) \bmod P(x)$.

2. $A(x) * B(x) = (x^2+1)(x+1) = x^3 + x + x^2 + 1$
   $C = x^3 + x^2 + x + 1 \equiv A(x) * B(x) \bmod P(x)$

The reduction polynomial is used to reduce $C(x)$ in order to reduce the result to $GF(2^4)$. Otherwise, a 'simple' multiplication without reduction would yield a result of a higher degree (e.g., with x^5) which would not belong to $GF(2^4)$ any more.

4.7

1. By the Extended Euclidean algorithm:

$$\begin{array}{lll} x^4+x+1 = [x^3](x)+[x+1] & t_2(x) = t_0 - q_1 t_1 = -q_1 = -x^3 = x^3 \\ x \quad\quad\quad\quad = [1](x+1)+1 & t_3(x) = t_1 - q_2 t_2 = 1 - 1 * x^3 = 1 - x^3 = x^3 + 1 \\ x+1 \quad\quad = [x+1](1)+0 & \end{array}$$

   So, $A^{-1} = x^3 + 1$.
   Check: $x * (x^3+1) = x^4 + x \equiv (x+1) + x \bmod P(x) = 1 \bmod P(x)$.

2. By the Extended Euclidean algorithm:

$$\begin{array}{ll} x^4+x+1 = [x^2+x+1](x^2+x)+[1] & t_2 = t_0 - q_1 t_1 = -q_1 = x^2+x+1 \\ x^2+x \quad\quad = [x^2+x]1+[0] & \end{array}$$

   So, $A^{-1} = x^2 + x + 1$.
   Check: $(x^2+x)(x^2+x+1) = x^4 + 2x^3 + 2x^2 + x = x^4 + x \equiv (x+1) + x \bmod P(x) = 1 \bmod P(x)$.

4.9

1. $A = 01_h, A(x) = 1$
   $A^{-1}(x) = 1 = 01_h$
   $A^{-1}(x)$ is now the input to the affine transformation of Rijndael as described in Section 4.2.1 of the Rijndael Specifications:

$$M \cdot A^{-1} + V$$

   where M and V are a fixed matrix and vector, respectively.

$$M \cdot A^{-1} + V = M \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

ByteSub(01_h) = $7C_h$

2. $A = 12_h$, $A(x) = x^4 + x$
   Apply extended Euclidean algorithm: $A^{-1}(x) = x^7 + x^5 + x^3 + x = AA_h$.

$$M \cdot A^{-1} + V = M \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

   Remark: It is (big) coincidence that $M \cdot A^{-1} = A^{-1}$. This only holds for this specific value of A^{-1}.
   ByteSub(12_h) = $C9_h$

4.11

■

$$B = ByteSub(A) = \begin{bmatrix} 16\ 16\ 16\ 16 \\ 16\ 16\ 16\ 16 \\ 16\ 16\ 16\ 16 \\ 16\ 16\ 16\ 16 \end{bmatrix}$$

- The ShiftRows operation does not change anything since all bytes of B equal each other.
- The MixColumn operation is equal for every resultig byte C_i and is described by $(01 + 01 + 02 + 03)_{hex} \cdot (16)_{hex}$. We have to remind, that all calculations have to be done in $GF(2^8)$, so that $(01 + 01 + 02 + 03)_{hex} = (01)_{hex}$ and hence, all resulting bytes of C equal to $(01)_{hex} \cdot (16)_{hex} = (16)_{hex} \Rightarrow$

$$C = MixColumn(B) = \begin{bmatrix} 16\ 16\ 16\ 16 \\ 16\ 16\ 16\ 16 \\ 16\ 16\ 16\ 16 \\ 16\ 16\ 16\ 16 \end{bmatrix}$$

- The first round key consists of 128 ones. So, the output of the first is

$$C \oplus K = \begin{bmatrix} 16\ 16\ 16\ 16 \\ 16\ 16\ 16\ 16 \\ 16\ 16\ 16\ 16 \\ 16\ 16\ 16\ 16 \end{bmatrix} \oplus \begin{bmatrix} FF\ FF\ FF\ FF \\ FF\ FF\ FF\ FF \\ FF\ FF\ FF\ FF \\ FF\ FF\ FF\ FF \end{bmatrix} = \begin{bmatrix} E9\ E9\ E9\ E9 \\ E9\ E9\ E9\ E9 \\ E9\ E9\ E9\ E9 \\ E9\ E9\ E9\ E9 \end{bmatrix}$$

4.13

$$\text{AES key} = \begin{bmatrix} 00 & 44 & 88 & 22 \\ 11 & 55 & 99 & 33 \\ 22 & 66 & 00 & 44 \\ 33 & 77 & 11 & 55 \end{bmatrix}$$

$$\text{round key } k_0 = \begin{bmatrix} 00 & 44 & 88 & 22 \\ 11 & 55 & 99 & 33 \\ 22 & 66 & 00 & 44 \\ 33 & 77 & 11 & 55 \end{bmatrix}$$

$$\text{round key } k_1 = \begin{bmatrix} c2 & 86 & 0e & 2c \\ 0a & 5f & c6 & f5 \\ de & b8 & b8 & fc \\ a0 & d7 & c6 & 93 \end{bmatrix}$$

$$\text{State after Inverse Byte Substitution (round 9)} = \begin{bmatrix} 77 & e9 & 45 & 58 \\ c4 & cf & fa & c3 \\ b8 & 66 & 87 & 95 \\ 05 & 23 & 31 & 20 \end{bmatrix}$$

$$\text{State after Key Addition Layer with } k_1 = \begin{bmatrix} b5 & 6f & 4b & 74 \\ ce & 90 & 3c & 36 \\ 66 & de & 3f & 69 \\ a5 & f4 & f7 & b3 \end{bmatrix}$$

$$\text{State after Inverse MixColumns Layer} = \begin{bmatrix} 2d & 5f & 5e & b0 \\ 3d & a0 & 0b & 9e \\ 95 & 5f & 4f & 86 \\ 3d & 75 & a5 & 30 \end{bmatrix}$$

$$\text{State after Inverse ShiftRows Layer} = \begin{bmatrix} 2d & 5f & 5e & b0 \\ 9e & 3d & a0 & 0b \\ 4f & 86 & 95 & 5f \\ 75 & a5 & 30 & 3d \end{bmatrix}$$

$$\text{State after Inverse Byte Substitution} = \begin{bmatrix} fa & 84 & 9d & fc \\ df & 8b & 47 & 9e \\ 92 & dc & ad & 84 \\ 3f & 29 & 08 & 8b \end{bmatrix}$$

$$\text{State after Key Addition Layer with } k_0 = \begin{bmatrix} fa & c0 & 15 & de \\ ce & de & de & ad \\ b0 & ba & ad & c0 \\ 0c & 5e & 19 & de \end{bmatrix}$$

$$\text{AES plaintext} = \begin{bmatrix} fa & c0 & 15 & de \\ ce & de & de & ad \\ b0 & ba & ad & c0 \\ 0c & 5e & 19 & de \end{bmatrix}$$

4.15

1. Without loss of generality we consider the leftmost MixColumn operation. Let's assume the 4 inputs are the byte B, which can have any value from $GF(2^8)$. The MixColumn is now:

$$\begin{pmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{pmatrix} = \begin{pmatrix} 02\ 03\ 01\ 01 \\ 01\ 02\ 03\ 01 \\ 01\ 01\ 02\ 03 \\ 03\ 01\ 01\ 02 \end{pmatrix} \begin{pmatrix} B \\ B \\ B \\ B \end{pmatrix}.$$

   We note that the computation of all four output bytes is identical and results in the same value:

$$C_i = (01+01+02+03)B \quad , \quad i = 0,1,2,3$$

   We write the constants 01, 02 and 03 now in their polynomial representation:

$$C_i = (1+1+x+(x+1))B \quad , \quad i = 0,1,2,3$$

   which yields

$$C_i = 1B = B \quad , \quad i = 0,1,2,3$$

2. The four input bytes are represented by 32 bits so that there is a total of 2^{32} different input values. There are exactly 256 of those 32-bit inputs where all four bytes have the same value, namely $(00,00,00,00)$, $(01,01,01,01)$, ..., (FF,FF,FF,FF). If the 2^{32} inputs are uniformly distributed (which holds if all input bytes are random), the probability that the four bytes are identical is, thus:

$$P(\text{4 identical input bytes}) = 256/2^{32} = 2^8/2^{32} = 2^{-24} = 1/16\,777\,216$$

3. Let's first look at the 1st round of AES, cf. Figs. 4.2 and 4.3. The four input bytes of any of the MixColumn boxes are only identical if the corresponding bytes prior to the four S-boxes are identical e.g., the values A_0, A_5, A_{10}, and A_{15} in Fig. 4.3. However, all bytes A_i are computed as the XOR of a plaintext byte with a key byte. If the key is chosen randomly, the four bytes A_i will also take random values, independently of of the plaintext. Hence, there is the small probability of

$$2^{-24}$$

   that any of the MixColumn boxes in the first round will have four identical inputs. The same argument holds in all subsequent rounds $2,3,...,n_r$, cf. Fig. 4.2. The 4 inputs to the MixColumn operation are only identical if the corresponding four bytes that are inputs to the preceding S-Boxes are identical. Since the S-box inputs are the result of an XOR operation which involves the subkey of the preceding round, the S-box inputs are also random. Hence, for all other MixColumn operations, the probability that all 4 input bytes are identical is also 2^{-24}.

4.17

1. $d = 01, b = 1 * (b_7x^7 + \ldots + b_0) = b.$
   $d_0 = b_0, d_1 = b_1, \ldots, d_7 = b_7.$
2. $d = 02 * b = x(b_7x^7 + \ldots + b_0) = b_7x^8 + b_6x^7 + \ldots + b_0x$
   $x^8 \equiv x^4 + x^3 + x + 1 \bmod P(x).$
   $d = b_6x^7 + b_5x^6 + b_4x^5 + [b_3 + b_7]x^4 + [b_2 + b_7]x^3 + b_1x^2 + [b_0 + b_7]x + b_7$
   $d_7 = b_6 \qquad d_6 = b_5$
   $d_5 = b_4 \qquad d_4 = b_3 + b_7$
   $d_3 = b_2 + b_7 \quad d_2 = b_1$
   $d_1 = b_0 + b_7 \quad d_0 = b_7$

3. $d = 03 * b = (x+1)b = xb + b$
   Using solutions from above:
   $d = (b_6 + b_7)x^7 + (b_5 + b_6)x^6 + (b_4 + b_5)x^5 + (b_3 + b_4 + b_7)x^4 + (b_2 + b_3 + b_7)x^3 + (b_1 + b_2)x^2 + (b_0 + b_1 + b_7)x + (b_0 + b_7)$
   $d_7 = b_6 + b_7 \qquad d_6 = b_5 + b_6$
   $d_5 = b_4 + b_5 \qquad d_4 = b_3 + b_4 + b_7$
   $d_3 = b_2 + b_3 + b_7 \quad d_2 = b_1 + b_2$
   $d_1 = b_0 + b_1 + b_7 \quad d_0 = b_0 + b_7$

4.19

1. $RC[8] = x^7 = (10000000)_2$
2. $RC[9] = x^8 = x^4 + x^3 + x + 1 = (00011011)_2$
3. $RC[10] = x^9 = x^8 \cdot x = x^5 + x^4 + x^2 + x = (00110110)_2$

Problems of Chapter 5

5.1

1. ECB

$$x = 01101\ 11011\ 11010\ 00110$$
$$y = 11001\ 11110\ 10110\ 00101$$

2. CBC and $IV = 11001$.

$$\begin{aligned} x &= 01101\ 11011\ 11010\ 00110 \\ y_1 &= e(x_1 \oplus IV) = e(01101 \oplus 11001) = e(10100) = 00011 \\ y_2 &= e(x_2 \oplus y_1) = e(11011 \oplus 00011) = e(11000) = 10010 \\ y_3 &= e(x_3 \oplus y_2) = e(11010 \oplus 10010) = e(01000) = 10000 \\ y_4 &= e(x_4 \oplus y_3) = e(00110 \oplus 10000) = e(10110) = 00111 \\ \Rightarrow y &= 00011\ 10010\ 10000\ 00111 \end{aligned}$$

3. CFB and $IV = 11001$.

$$\begin{aligned} x &= 01101\ 11011\ 11010\ 00110 \\ y_1 &= e(IV) \oplus x_1 = e(11001) \oplus 01101 = 11010 \oplus 01101 = 10111 \\ y_2 &= e(y_1) \oplus x_2 = e(10111) \oplus 11011 = 01111 \oplus 11011 = 10100 \\ y_3 &= e(y_2) \oplus x_3 = e(10100) \oplus 11010 = 00011 \oplus 11010 = 11001 \\ y_4 &= e(y_3) \oplus x_4 = e(11001) \oplus 00110 = 11010 \oplus 00110 = 11100 \\ \Rightarrow y &= 10111\ 10100\ 11001\ 11100 \end{aligned}$$

4. OFB and $IV = 11001$.

$$\begin{aligned} x &= 01101\ 11011\ 11010\ 00110 \\ s_1 &= e(IV) = e(11001) = 11010 & y_1 &= s_1 \oplus x_1 = 11010 \oplus 01101 = 10111 \\ s_2 &= e(s_1) = e(11010) = 10110 & y_2 &= s_2 \oplus x_2 = 10110 \oplus 11011 = 01101 \\ s_3 &= e(s_2) = e(10110) = 00111 & y_3 &= s_3 \oplus x_3 = 00111 \oplus 11010 = 11101 \\ s_4 &= e(s_3) = e(00111) = 01101 & y_4 &= s_4 \oplus x_4 = 01101 \oplus 00110 = 01011 \\ \Rightarrow y &= 10111\ 01101\ 11101\ 01011 \end{aligned}$$

5. CTR and $IV = 11001$.

$$
\begin{aligned}
x &= 101\,100\,011\,000 \\
y_1 &= e(IV+0) \oplus x_1 = e(11001) \oplus x_1 = 11010 \oplus 01101 = 10111 \\
y_2 &= e(IV+1) \oplus x_2 = e(11010) \oplus x_2 = 10110 \oplus 11011 = 01101 \\
y_3 &= e(IV+2) \oplus x_3 = e(11011) \oplus x_3 = 11110 \oplus 11010 = 00100 \\
y_4 &= e(IV+3) \oplus x_4 = e(11100) \oplus x_4 = 10011 \oplus 00110 = 10101 \\
\Rightarrow y &= 10111\,01101\,00100\,10101
\end{aligned}
$$

5.3

The decryption of an "CBC-encrypted" file is defined by $x_i = d_K(y_i) \oplus y_{i-1}$. Since you know the key K and the pair (x_0, y_0) (from the first file), the unknown IV can easily be obtained by converting the equation:

$$IV = y_{-1} = d_k(y_0) \oplus x_0$$

After that, the second (unidentified) file can easily be decrypted by using the decryption equation mentioned above (with $y_{-1} = IV$).

5.5

If the same IV is used for the OFB encryption, the confidentiality may be compromized. If a plaintext block x_j of such a message m is known, the output can be computed easily from the ciphertext block y_j of the message m. This information then allows the computation of the plaintext block x'_j of any other message m' that is encrypted using the same IV.

5.7

1.
2. The problem with the scheme is that there are only 256 different inputs FB_i to the AES algorithm. That means there are only 256 different output vectors of length 128bit that form the keystream. To make things worse, the cipher output will run into a cycle quickly. Let's denote the sequence of feedback bytes by $FB_1, FB_2, \ldots$ As soon as a feedback byte FB_j is generated that is equal to an earlier one FB_i, i.e., $i < j$, the sequence

   $$FB_i, FB_{i+1}, \ldots, FB_j = FB_i, FB_{i+1}, \ldots, FB_j = FB_i, FB_{i+1}, \ldots$$

   repeats periodically. Since there are only 256 different values for FB, the maximum sequence length is 256. Since each value is associated with a 128 (16 byte) AES output, the keystream sequence s_i has a maximum cycle length of:

   $$128 \times 16 = 2048\text{byte} = 2\text{kbyte}.$$

   After this, the stream cipher output must repeat (and odds are that the cycle lenght is much shorter). Thus, if an attacker has to know at most 2kB of plaintext in order to recover the entire stream cipher output with which he can decrypt all other ciphertext.

3. No, we still only generate a maximum of 256 keystream words of length 16 byte.

Remark: In the chapter on hash functions we will learn about the birthday paradox. This is applicable here too and tells us that the expected length of the sequence is in fact approximately $\sqrt{2}256 = 16$.

5.9
The counter has to encrypt 1 TB of data without repeating itself. This yields an IV of maximum size of $91 = 128 - 36$ bits.

5.11
A missing or deleted bit in y_i affects the i-th feedback bit which enters the shift register of size of κ bit. After $\kappa + 1$ steps, the affected feedback bit leaves the shift register. As a consequence, all subsequent decryptions (i.e., decryptions of $y_{i+\kappa+}...$) are again correct.

5.13

1. $y = e_{k_3}(e_{k_2}^{-1}(e_{k_1}(x)))$
   $x = e_{k_1}^{-1}(e_{k_2}(e_{k_3}^{-1}(y)))$
2. Compared to a triple encryption, the EDE mode does not increase the security level. However, it allows for backward compatibility if we use the same key for encryption and decryption.
3. Brute-force complexity: $(2^\kappa)^3 = (2^{80})^3 = 2^{240}$
   MitM memory: $2^\kappa = 2^{80}$
   MitM computationaly complexity: $(2^\kappa)^2 = (2^{80})^2 = 2^{160}$
   $\Rightarrow$ effective key length: 160
   Even with todays' computers, such an attack is infeasible.
4. In general: $2^{l\cdot\kappa - t\cdot n}$ with t pairs of plain and ciphertext and l as the number of encryptions.
   $\Rightarrow$ check for $t = 1, 2, 3, \ldots$
   $\Rightarrow$ For $t = 4$ we have a probability for a wrong key of $2^{3\cdot 80 - 4\cdot 64} = 2^{-16} < 0$

5.15
We can use the following formula to estimate the success probabilities:

$$Pr(K' \neq K) = 2^{l\,k - t\,n},$$

with l being the number of encryptions, k the key length, t the number of available plaintext/ciphertext blocks and n as the block length.

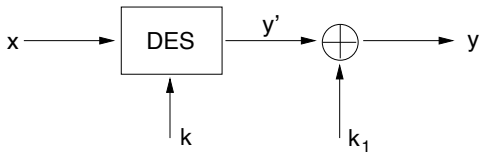
1. The desired error probability is given by $Pr(K' \neq K) < 2^{-20}$: Using above formula, we find that it is fulfilled for $t \geq 5$.
2. Re-ordering the formula to

$$k = \frac{\log_2(Pr(K' \neq K)) + t\,n}{l}$$

and using $l = 2$, $n = 80$ and $t = 5$, leads to a maximum key length of $k = 195$ bits.

3. With the parameters available, we get the expression $t\,n = 512$, which is equal to $k\,l = 512$. Hence, the correct keys k_1, k_2 cannot uniquely be determined, which follows from the error probability of $Pr(K' \neq K) = 1$.

5.17 Notation: $DESA_{k,k_1}(x) = DES_k(x) \oplus k_1 = y$
input: 2 pairs of known plaintext/ciphertext $(x_1, y_1), (x_2, y_2)$ such that:



$$DESA_{k,k_1}(x_i) = y_i \quad ; \quad i = 1,2$$

$$\begin{aligned} y_1 \oplus y_2 &= (DES_k(x_1) \oplus k_1) \oplus (DES_k(x_2) \oplus k_1) \\ &= DES_k(x_1) \oplus DES_k(x_2) \qquad (14.2) \\ &= y_1' \oplus y_2' \qquad (14.3) \end{aligned}$$

Key Search Algorithm (2 steps):

1. Find Key k:
   $DES_{k_i}(x_1) \oplus DES_{k_i}(x_2) \stackrel{?}{=} y_1 \oplus y_2 \quad i = 0,1,2,\ldots$ [works because of Equation (14.2)]

2. Find Key k_1:
   $y_1' = DES_k(x_1)$ [use key k from Step 1]
   $k_1 = y_1' \oplus y$

Problems of Chapter 6

6.1 From a theoretical point of view, public key cryptography can be used as a replacement for symmetric cryptography. However, in practical applications, symmetric ciphers tend to be approximately 1000 times faster than public key schemes. Hence, symmetric ciphers are used when it comes to bulk data encryption.

6.3 If every pair out of $n = 120$ employees requires a distinct key, we need in sum

$$n \cdot \frac{n-1}{2} = 120 \cdot \frac{120-1}{2} = 7140$$

key pairs. Remark that each of these key pairs have to be exchanged in a secure way (over a secure channel)!

6.5

1. $\gcd(7469, 2464) = 77$
2. $\gcd(4001, 2689) = 1$
3. $\gcd(333200, 286875) = 425$

6.7

1. $\gcd(26, 7) = 1$
   $q_1 = 3$, $q_2 = 1$, $q_3 = 2$
   $t_2 = -3$, $t_3 = 4$, $t_4 = -11$
   $a^{-1} \equiv t_4 \bmod m \equiv -11 \bmod 26 = 15$
2. $\gcd(999, 19) = 1$
   $q_1 = 52$, $q_2 = 1$, $q_3 = 1$, $q_4 = 2$, $q_5 = 1$
   $t_2 = -52$, $t_3 = 53$, $t_4 = -105$, $t_5 = 263$, $t_6 = -368$
   $a^{-1} \equiv t_6 \bmod m \equiv -368 \bmod 999 = 631$

6.9

1. $\phi(p) = (p^1 - p^0) = p - 1$
2. $\phi(p \cdot q) = (p-1) \cdot (q-1)$
   $\phi(15) = \phi(3 \cdot 5) = 2 \cdot 4 = 8$
   $\phi(26) = \phi(2 \cdot 13) = 1 \cdot 12 = 12$

6.11

1. $m = 6$; $\phi(6) = (3-1) \cdot (2-1) = 2$;
   Euler's Theorem: $a^2 \equiv 1 \bmod 6$, if $\gcd(a, 6) = 1$
   $0^2 \equiv 0 \bmod 6$;
   $1^2 \equiv 1 \bmod 6$;
   $2^2 \equiv 4 \bmod 6$;
   $3^2 \equiv 9 \equiv 3 \bmod 6$;
   $4^2 \equiv 16 \equiv 4 \bmod 6$;
   $5^2 \equiv 25 \equiv 1 \bmod 6$

2. $m = 9$; $\phi(9) = 3^2 - 3^1 = 9 - 3 = 6$;
   Euler's Theorem: $a^6 \equiv 1 \bmod 9$, if $\gcd(a,9) = 1$
   $0^6 \equiv 0 \bmod 9$;
   $1^6 \equiv 1 \bmod 9$;
   $2^6 \equiv 64 \equiv 1 \bmod 9$;
   $3^6 \equiv (3^3)^2 \equiv 0^2 \equiv 0 \bmod 9$;
   $4^6 \equiv (2^6)^2 \equiv 1^2 \equiv 1 \bmod 9$;
   $5^6 \equiv 1 \bmod 9$;
   $6^6 \equiv 2^6 \cdot 3^6 \equiv 1 \cdot 0 \equiv 0 \bmod 9$;
   $7^6 \equiv 1 \bmod 9$;
   $8^6 \equiv 1 \bmod 9$

6.13
Euclid's Algorithm:

$$\text{Iteration 2: } r_0 = q_1 r_1 + r_2 \qquad r_2 = r_0 - q_1 r_1 = s_2 r_0 + t_2 r_1 \tag{1}$$
$$\text{Iteration 3: } r_1 = q_2 r_2 + r_3 \qquad r_3 = [-q_2] \cdot r_0 + [1 + q_1 q_2] \cdot r1 = s_3 r_0 + t_3 r_1 \tag{2}$$

$$\Rightarrow \text{from (1),(2): } s_2 = 1; \quad s_3 = -q_2 \tag{3}$$
$$t_2 = -q_1; t_3 = 1 + q_1 q_2 \tag{4}$$

The iteration formula for the Euclidean Algorithm gives:

(5) $s_2 \overset{EA}{=} s_0 - q_1 s_1 \overset{(3)}{=} 1$

(6) $s_3 \overset{EA}{=} s_1 - q_2 s_2 \overset{(3)}{=} s_1 - q_2 \overset{(3)}{=} -q_2$
$\overset{(6)}{\Rightarrow} s_1 = 0 \qquad \overset{(5)}{\Rightarrow} s_0 = 1$

(7) $t_2 \overset{EA}{=} t_0 - q_1 t_1 \overset{(4)}{=} -q_1$

(8) $t_3 \overset{EA}{=} t_1 - q_2 t_2 \overset{(4)}{=} t_1 + q_1 q_2 \overset{(4)}{=} 1 + q_1 q_2$
$\overset{(8)}{\Rightarrow} t_1 = 1 \qquad \overset{(7)}{\Rightarrow} t_0 = 0$

Problems of Chapter 7

7.1

1. Only $e = 31$ is a valid public key, because $\Phi(n) = (p-1)(q-1) = 40 \cdot 16 = 640 = 2^7 \cdot 5$. Furthermore $gcd(e_i, \phi(n)) = 1$ has to be fulfilled. Hence, only $e_2 = 49$ may be used as public exponent.
2. $K_{pub} = (n,e) = (697,49)$
   Calculation of $d = e^{-1} \bmod \phi(n) = 49^{-1} \bmod 640$ using EEA:

$$\begin{aligned} 640 &= 13 \cdot 49 + 3 \\ 49 &= 16 \cdot 3 + 1 \\ \Leftrightarrow 1 &= 49 - 16 \cdot 3 \\ &= 49 - 16(640 - 13 \cdot 49) \\ &= 209 \cdot 49 - 16 \cdot 640 \\ \Rightarrow \quad 49^{-1} &\bmod 640 \equiv 209. \end{aligned}$$

So, the private key is defined by $K_{pr} = (p,q,d) = (41,17,209)$.

7.3

1. $e = 3$; $y = 26$
2. $d = 27$; $y = 14$

7.5

1. In this case, a brute-force attack on all possible exponents would be easily feasible.
2. Wiener's attack is applicable if $d < 1/3\, n^{1/4}$. If we ignore the factor of $1/3$, d must have at least $2048/4 = 512$ bits.
3. It holds
$$e \cdot d \equiv 1 \mod \phi(n)$$
and thus:
$$e \cdot d > \phi(n)$$
from which follows:
$$d > \phi(n)/e.$$
In order to get the bit length of a number x we have to compute its binary logarithm: $|x| = \lceil \log_2(x) \rceil$. We apply this to both sides of the equation above:
$$\begin{aligned} \lceil \log_2(d) \rceil &> \lceil \log_2(\phi(n)/e) \rceil \\ \lceil \log_2(d) \rceil &> \lceil \log_2(\phi(n)) \rceil - \lceil \log_2(e) \rceil \\ |d| &> |\phi(n)| - |e| \end{aligned}$$
We now show that $|\phi(n)| = |n|$. It holds
$$\phi(n) = (p-1)(q-1) = pq - p - q - 1 = n - p - q - 1.$$
Crucially, n has **twice** the bit length of p as well as of q, i.e., we are subtracting relatively small numbers from n. Thus, it is almost always the case that $n - p - q - 1$ has the same bit length than n itself.

7.7

$p = 31$, $q = 37$, $e = 17$, $y = 2$

- $n = 31 \cdot 37 = 1147$
  $d = 17^{-1} = 953 \bmod 1080$
- $d_p = 953 \equiv 23 \bmod 30$
  $d_q = 953 \equiv 17 \bmod 36$
- $x_p = y^{d_p} = 2^{23} \equiv 8 \bmod 31$
  $x_q = y^{d_q} = 2^{17} \equiv 18 \bmod 37$
- $c_p = q^{-1} = 37^{-1} \equiv 6^{-1} \equiv 26 \bmod 31$
  $c_q = p^{-1} = 31^{-1} \equiv 6 \bmod 37$
- $x = [qc_p]x_p + [pc_q]x_q =$
  $[37 \cdot 26]8 + [31 \cdot 6]18 =$
  $8440 = 721 \bmod 1147$

7.9

Alice		**Bob**
		setup: $k_{pr} = d$; $k_{pub} = e$ publish e, n
choose random session key k_{ses} $y = e_{k_{pub}}(k_{ses}) = k_{ses}^e \bmod n$	$\xrightarrow{y}$	
		$k_{ses} = d_{k_{pr}}(y) = y^d \bmod n$

Alice completely determines the choice of the session key k_{ses}.

Note that in practice k_{ses} might be much longer than needed for a symmmetric-key algorithm. For instance, k_{ses} may have 1024 bits but only 128 actual key bits are needed. In this case just use the 128 MSB (or LSB) bits are used and the remaining bits are discarded. Often, it is safe practice to apply a cryptographic hash function first to k_{ses} and then take the MSB or LSB bits.

7.11

1. Encryption equation: $y \equiv x^e \bmod n$. We cannot solve the equation analytical, because the exponentiation takes place in a finite ring, where no efficient algorithms for computing roots is known.
2. $$\Phi(n) = p \cdot q$$
   No! The calculation of $\Phi(n)$ presumes the knowledge of p and q, which we do not have.
3. Factorization yields: $p = 43$ and $q = 61$
   $\Phi(n) = 42 \cdot 60 = 2520$
   $d \equiv e^{-1} \bmod 2520 \equiv 191$
   $x = 1088$

7.13

1. A message consists of, let's say, m pieces of ciphertext $y_0, y_1, \ldots, y_{m-1}$. However, the plaintext space is restricted to 95 possible values and the ciphertext space too.

That means we only have to test 95 possible plaintext characters to build up a table containing all possible ciphertext characters:

Test: $y_i \stackrel{?}{\equiv} j^e \bmod n;\ j = 32, 33, \ldots, 126$

2. SIMPSONS
3. With OAEP padding a random string *seed* is used with every encryption. Since *seed* has in practice a length of 128–160 bits, there exist many, many different ciphertexts for a given plaintext.

7.15

The basic idea is to represent the exponent in a radix 2^k representation. That means we group k bits of the exponent together. The first step of the algorithm is to precompute a look-up table with the values $A^0 = 1$, $A^1 = A$, A^2, …, A^{2^k-1}. Note that the exponents of the look-up table values represent all possible bit patterns of length k. The table computation requires $2^k - 2$ multiplications (note that computing A^0 and A^1 is for free). After the look-up table has been computed, the two elementary operations in the algorithm are now:

- Shift intermediate exponent by k positions to the left by performing k subsequent squarings (Recall: The standard s-a-m algorithm shifts the exponent only by one position by performing one squaring per iteration.)
- The exponent has now k trailing zeros at the rightmost bit positions. Fill in the required bit pattern for the exponent by multiplying the corresponding value from the look-up table with the intermediate result.

This iteration is only performed l/k times, where $l+1$ is the bit length of the exponent. Hence, there are only l/k multiplications being performed in this part of the algorithm.

An exact description of the algorithm, which is often referred to as *k-ary exponentiation*, is given in [189]. Note that the bit length of the exponent in this description is $t\,k$ bits. An example for the case $k = 3$ is given below.

The complexity of the algorithm for an $l+1$-bit exponent is $2^k - 3$ multiplications in the precomputation phase, and about $l - 1$ squarings and $l(2^k - 1)/2^k$ multiplications in the main loop.

Example 14.4. The goal is to compute $g^e \bmod n$ with k-ary where $n = 163$, $g = 12$, $k = 3$, $e = 145_{10} = 221_{8=2^3} = 10\,010\,001_2$

Precomputation:

$g_0 := 1$
$g_1 := 12$
$g_2 := g_1 \cdot 12 = 144$
$g_3 := g_2 \cdot 12 = 1728 \bmod 163 = 98$
$g_4 := g_3 \cdot 12 = 1176 \bmod 163 = 35$
$g_5 := g_4 \cdot 12 = 420 \bmod 163 = 94$
$g_6 := g_5 \cdot 12 = 1128 \bmod 163 = 150$
$g_7 := g_6 \cdot 12 = 1800 \bmod 163 = 7$

Exponentiation:

Iteration	Exponent (base 2)	Calculation	Operation
0	10	$A := g_2 = 144$	TLU
1a	10 000	$A := A^8 \bmod 163 = 47$	3 SQ
1b	10 010	$A := A \cdot g_2 = 6768 \bmod 163 = 85$	MUL
2a	10 010 000	$A := A^8 \bmod 163 = 140$	3 SQ
2b	10 010 001	$A := A \cdot g_1 = 1680 \bmod 163 = 50$	MUL

In each iteration, three squarings results in a left shift which makes space for multiplying by the appropriate precomputed power of g. For instance, if the next binary digits to be processed are $(010)_2 = (\mathbf{2})_{10}$, we take the value $g_{\mathbf{2}} = g^2$ from the look-up-table and multiply it by the intermediate result.

This example emphasizes the impact of the precomputations on the efficiency of the k-ary algorithm: For the small operand lengths used here, the overall cost for the exponentiation is worse than for the s-a-m algorithm. This changes a lot for real-world operands with 1024 or more bits, as the size of the look-up-table only depends on the window size k, and not on the operand length.

◇

7.19

$$\text{2048-bit RSA}: \quad P(\text{odd number is a prime}) \approx \frac{2}{\ln(2^{1024})} = \frac{2}{1024\ln(2)} \approx 0.00228$$

$$\text{3072-bit RSA}: \quad P(\text{odd number is a prime}) \approx \frac{2}{\ln(2^{1536})} = \frac{2}{1536\ln(2)} \approx 0.00188$$

$$\text{4096-bit RSA}: \quad P(\text{odd number is a prime}) \approx \frac{2}{\ln(2^{2048})} = \frac{2}{2048\ln(2)} \approx 0.000141$$

Problems of Chapter 8

8.1

1. $\mathbb{Z}_5^*$:

a	1	2	3	4
ord(a)	1	4	4	2

2. $\mathbb{Z}_7^*$:

a	1	2	3	4	5	6
ord(a)	1	3	6	3	6	2

3. $\mathbb{Z}_{13}^*$:

a	1	2	3	4	5	6	7	8	9	10	11	12
ord(a)	1	12	3	6	4	12	12	4	3	6	12	2

8.3

1.
$|\mathbb{Z}_5^*| = 4$
$|\mathbb{Z}_7^*| = 6$
$|\mathbb{Z}_{13}^*| = 12$

2. yes
3.
$\mathbb{Z}_5^* : 2,3$
$\mathbb{Z}_7^* : 3,5$
$\mathbb{Z}_{13}^* : 2,6,7,11$

4.
$\phi(4) = 2$
$\phi(6) = 2$
$\phi(12) = 4$

8.5

1. $K_{pub_A} = 8 \qquad K_{pub_B} = 32 \qquad K_{AB} = 78$
2. $K_{pub_A} = 137 \qquad K_{pub_B} = 84 \qquad K_{AB} = 90$
3. $K_{pub_A} = 394 \qquad K_{pub_B} = 313 \qquad K_{AB} = 206$

8.7

Both values would yield public keys that would immediately allow to recognize the private key. If the private key is equal to 1, the public key would be identical to the primitive element α. If an attacker would detect this identity, he would know that $k_{pr} = 1$. If the private key is equal to $p-1$, the public key would take the value 1 according to Fermat's Little Theorem. If an attacker notices this, he can deduce that $k_{pr} = p-1$.

8.9

1. The order of $a = p-1$ is 2, since

$$a^1 = a = p-1; \quad a^2 = (p-1)^2 \equiv (-1)^2 = 1.$$

2. The subgroup H_a, which is generated by a is defined by $H_a = \{1, p-1\}$ (or equally $H_a = \{1, -1\}$).
3. An attacker could alter the mutually used element a to an element a' of the previously mentioned form, so that it generates a subgroup with only two elements. Hence, the Diffie–Hellman key exchange can only yield in two different key and the attacker only has two test both possibilities to determine the right key.

8.11

Alice		**Oscar**		**Bob**
$\alpha^{a_A} = b_A$	$\xrightarrow{b_A}$	$\alpha^{a_O} = b_O$	$\xrightarrow{b_O}$	
	$\xleftarrow{b_O}$		$\xleftarrow{b_B}$	$\alpha^{a_B} = b_B$
$b_O^{a_A} = \alpha^{a_A \cdot a_O} = K_{AO}$		$b_A^{a_O} = \alpha^{a_A \cdot a_O} = K_{AO}$ $b_B^{a_O} = \alpha^{a_B \cdot a_O} = K_{BO}$		$b_O^{a_B} = \alpha^{a_B \cdot a_O} = K_{BO}$

Oscar shares now a *secret* key with Alice and Bob. Alice and Bob both don't know about it and think they share a key with each other. Oscar can now decrypt, read, and encrypt any messages between Alice and Bob without them learning about it if he continues to intercept all encrypted messages.

This is the infamous *man-in-the-middle* attack. This attack is, in essence, responsible for things such as certificates, public-key infrastructures, etc.

8.13
Compute β: $\beta = \alpha^d \bmod p$.
Encrypt: $(k_E, y) = (\alpha^i \bmod p, x \cdot \beta^i \bmod p)$.
Decrypt: $x = y(k_E^d)^{-1} \bmod p$.

1. $(k_E, y) = (29, 296)$, $x = 33$
2. $(k_E, y) = (125, 301)$, $x = 33$
3. $(k_E, y) = (80, 174)$, $x = 248$
4. $(k_E, y) = (320, 139)$, $x = 248$

8.15
Oscar knows x_n, y_n and n (by just counting the number of ciphertexts). The first step of a possible attack is to calculate

$$k_{M,n} = y_n \cdot x_n^{-1} \bmod p. \tag{14.4}$$

Caused by the previously mentioned PRNG, beginning with $k_{M,n-1}$, $k_{M,j-1}$ can easily calculated recursivley through

$$k_{M,j-1} = \beta^{i_{j-1}} = \beta^{i_j - f(j)} = \beta^{i_j} \cdot \beta^{-f(j)} = k_{M,j-1} \cdot \beta^{-f(j)} \bmod p \tag{14.5}$$

where the values of all variables are known. With the knowledge of $k_{M,j}$ for all j, Oscar is now able to decrypt the whole ciphertext by solving the usual decryption equation

$$x_j = y_j \cdot k_{M,j}^{-1} \bmod p \tag{14.6}$$

8.17

1. By choosing a different secret exponent i, the ciphertext y of the same plaintext x is different everytime. Even if a pair of plaintext/ciphertext is compromised, such a pair will most likely not repeat a second time in a non-deterministic encryption scheme!

2. In general, there are $\#\{2,3,\cdots,p-2\} = p-3$ different valid ciphertexts for a single plaintext. I.e., we have 464 different possibilities for $p = 467$.
3. The plain RSA cryptosystem is deterministic. A specific plaintext always yields the same ciphertext assuming the same public parameters.

8.19

1. $x = \{18,24,12,12,4,19,17,8,2\}$, $y_i = x_i^9 \bmod 29$, i.e.,
   $y = \{27,25,12,12,13,11,17,15,19\} = \{\mathtt{B},\mathtt{Z},\mathtt{M},\mathtt{M},\mathtt{N},\mathtt{L},\mathtt{R},\mathtt{P},\mathtt{T}\}$
2. $d = e^{-1} \bmod p-1 \equiv 9^{-1} \bmod 28 \equiv 25$
3. e can not be arbitrarily chosen since its inverse must exist, i.e., $\gcd(e, p-1) = 1$. Furthermore, an attacker shall not be able to compute the e-th root to reverse the exponentiation. Thus, e should be large enough.
4. No since the decryption key can be computed from the encryption key e and the prime p!
5. Knowing a pair of plaintext and ciphertext does not allow to compute the secret key e or d. The Pohlig-Hellman cipher is resistant against known-plaintext attacks.

8.21

1. with $k_{pr} = p-1$: $k_{pub} = \alpha^{p-1} \bmod p = \alpha^{p-1 \bmod p-1} \bmod p \equiv \alpha^0 \bmod p \equiv 1 => k_{pub} = 1$
2. $(p-1)^2 \bmod p = p^2 - 2p + 1 \bmod p \equiv 1 \Rightarrow \mathrm{ord}(p-1) = 2$
   $H_\alpha = \{1, p-1\}$

Problems of Chapter 9

9.1
$a = 2$, $b = 2$
$4 \cdot 2^3 + 27 \cdot 2^2 = 4 \cdot 8 + 27 \cdot 4 = 32 + 108 = 140 \equiv 4 \neq 0 \bmod 17$

9.3 $17 + 1 - 2\sqrt{17} \approx 9,75 \leq 19 \leq 17 + 1 + 2\sqrt{17} \approx 26,25$ q.e.d.

9.5

1. The points of E are

$$\{(0,3),(0,4),(2,3),(2,4),(4,1),(4,6),(5,3),(5,4)\}$$

2. The group order is given by

$$\#G = \#\{O,(0,3),(0,4),(2,3),(2,4),(4,1),(4,6),(5,3),(5,4)\} = 9$$

3. Compute all multiples of α:

$$
\begin{aligned}
0 \cdot \alpha &= O \\
1 \cdot \alpha &= (0,3) \\
2 \cdot \alpha &= (2,3) \\
3 \cdot \alpha &= (5,4) \\
4 \cdot \alpha &= (4,6) \\
5 \cdot \alpha &= (4,1) \\
6 \cdot \alpha &= (5,3) \\
7 \cdot \alpha &= (2,4) \\
8 \cdot \alpha &= (0,4) \\
9 \cdot \alpha &= O = 0 \cdot \alpha \\
&\Rightarrow \text{ord}(\alpha) = 9 = \#G \\
&\Rightarrow \alpha \text{ is primitive since it generates the group!}
\end{aligned}
$$

9.7 The element order divides the group order $\Longrightarrow$ possible orders of α are 1, 2, 4, 8, 16. We can exclude ord(α)= 1 since $\alpha \neq \mathcal{O}$. Thus, we only need to check if ord(α)= 2, ord(α)= 4, and ord(α)= 8 to see if α has the order 2, 4, 8, or 16. If $2\alpha = \mathcal{O}$ or $4\alpha = \mathcal{O}$ or $8\alpha = \mathcal{O}$, then ord(α)= 2 or 4, 8, respectively, else ord(α)= 16. Thus, the computation involves at most 3 point doublings.

9.9

1. Solution with help of a table:

i	i^2 mod 11	i^3+9i+1 mod 11
0	0	1
1	1	0
2	4	5
3	9	0
4	5	2
5	3	6
6	3	7
7	5	0
8	9	2
9	4	8
10	1	2

   Points on E are $E = \{(0,1),(0,10),(1,0),(2,4),(2,7),(3,0),(7,0),\mathcal{O}\}$.
   Thus, the order of the curve is given by $ord(E) = 8$.
2. 2P=(2,4)+(2,4):

$$\lambda = \frac{3 \cdot 2^2 + 9}{2 \cdot 4} = 10 \cdot 8^{-1} = 10 \cdot 7 = 4$$
$$x_3 = 4^2 - 2 - 2 = 1$$
$$y_3 = 4(2-1) - 4 = 0$$
$$2P = (1,0)$$

3P=(1,0)+(2,4):

$$\lambda = \frac{4-0}{2-1} = 4$$
$$x_3 = 4^2 - 1 - 2 = 2$$
$$y_3 = 4(1-2) - 0 = -4 = 7$$
$$3P = (2,7)$$

9.11

1. $9 \cdot P = (1001_{|2})P = (2 \cdot (2 \cdot (2 \cdot P))) + P = (4,10)$
2. $20 \cdot P = (10100_{|2})P = (2 \cdot (2 \cdot (2 \cdot (2 \cdot P) + P))) = (19,13)$

9.13

$K = aB = 6 \cdot B = 2(2B + B)$

$2B = (x_3, y_3) : x_1 = x_2 = 5; y_1 = y_2 = 9$
$s = (3x_1^2 + a) \cdot y_1^{-1} = (3 \cdot 25 + 1)(2 \cdot 9)^{-1} = 76 \cdot 18^{-1} \bmod 11$
$s \equiv 10 \cdot 8 = 80 \equiv 3 \bmod 11$
$x_3 = s^2 - x_1 - x_2 = 3^2 - 10 = -1 \equiv 10 \bmod 11$
$y_3 = s(x_1 - x_3) - y_1 = 3(5-10) - 9 = -15 - 9 = -24 \equiv 9 \bmod 11$
$2B = (10,9)$

$3B = 2B + B = (x_3', y_3') : x_1 = 10, x_2 = 5, y_1 = 9, y_2 = 9$
$s = (y_2 - y_1)(x_2 - x_1)^{-1} = 0 \bmod 11$
$x_3' = 0 - x_1 - x_2 = -15 \equiv 7 \bmod 11$
$y_3' = s(x_1 - x_3) - y_1 = -y_1 = -9 \equiv 2 \bmod 11$
$3B = (7,2)$

$6B = 2 \cdot 3B = (x_3'', y_3'') : x_1 = x_2 = 7, y_1 = y_2 = 2$
$s = (3x_1^2 + a) \cdot y_1^{-1} = (3 \cdot 49 + 1) \cdot 4^{-1} \equiv 5 \cdot 4^{-1} \equiv 5 \cdot 3 = 15 \equiv 4 \bmod 11$
$x_3'' = s^2 - x_1 - x_2 = 4^2 - 14 = 16 - 14 = 2 \bmod 11$
$y_3'' = s(x_1 - x_3) - y_1 = 4(7-2) - 2 = 20 - 2 = 18 \equiv 7 \bmod 11$
$6B = (2,7) \Rightarrow K_{AB} = 2$

9.15

A brute-force attack on a 128-bit key currently is computitional infeasible!
In this context, a much more efficient attack is to make use of the correlation between the $x-$ and $y-$ coordinate of a point. Since it is known that there is an inverse

for every point $P = (x, y)$ with $-P = (x, -y)$, it would be the easiest approach to test all 2^{64} possible $x-$coordinates by solving the curve equation. The effective key length is then reduced to 65 bits, which may be insufficient in a few years (if this problem has not already been broken by well-funded intelligence services).

Problems of Chapter 10

10.1

1. ■ If a message from Alice to Bob is found to be authentic, i.e., in fact originated from Alice, integrity is automatically assured, since an alteration by Oscar would make *him* the originator. However, this can't be the case if sender authenticity is assured.
   ■ No, a message can still be unaltered but message authenticity is not given. For instance, Oscar could masquerade as Alice and send a message to Bob saying that it is from Alice. Although the message ar rives unaltered at Bob's (integrity is thus assured) sender authenticity is not given.
2. No, for achieving confidentiality we have to use encryption. However, encrypted data can be altered during transmission without the receiving (or sending) party noticing.

10.3

1. $\phi(11111) = (271-1)\cdot(41-1) = 10800$
   Choose e such that gcd(e,10800) = 1
   $\Longrightarrow$ e=7
   $d = e^{-1} \equiv 1543 \bmod 10800$
   $1234^{1543} \equiv 8182 \bmod 11111$

2. The signature computation gets very efficient for small exponents e.

10.5

1. $6292^b \equiv x \bmod n \Rightarrow$ valid signature
2. $4768^b \neq x \bmod n \Rightarrow$ invalid signature
3. $1424^b \equiv x \bmod n \Rightarrow$ valid signature

10.7

Oscar receives the message, alters it and signs it with his own private key a'. Then he sends the new message together with the signature and the alleged public key (n', e') of Alice (which is instead the one of Oscar).

10.9

1. $sig_{K_{pr}}(x) = x^d \bmod n = y$
   $ver_{K_{pub}}(x, y): \quad x \stackrel{?}{\equiv} y^e \bmod n$
   Assume that d has l bits.

Using the square & multiply algorithm, the average signing will take:
$\#\otimes \approx l$ squarings + $\frac{1}{2} \cdot l$ multiplications = $\underline{\frac{3}{2} \cdot l}$
Since $b = 2^{16}+1 \equiv 65537 \equiv 1000000000000001_2$, the average verification takes:
$\#\otimes = 16\ squarings + 1\ multiplication = 17$

2. Signing takes longer than verification.
3.

$l\ [bits]$	T_{unit}	n	$\frac{T_{unit}}{operation}$	$T(sig)$	$T(ver)$
1024	100 ns	32	102.4 μs	157.3 ms	1.741 ms
2048	100 ns	64	409.6 μs	1.258 s	6.963 ms

4. $T(1\otimes) = (\frac{l}{32})^2 \cdot \frac{1}{f}$; time for one multiplication modulo p

$$T(sig) = \frac{3}{2} \cdot l \cdot \frac{T_{unit}}{operation} = 0.5\ s$$

$$\frac{T_{unit}}{operation} = (\frac{l}{32})^2 \cdot T_{unit}$$

$$F \geq \frac{1}{T_{unit}}[Hz]$$

i) $\underline{50.33\ MHz}$
ii) $\underline{402.6\ MHz}$

10.11

1. $\alpha^x = 3^{10} \equiv 25 \bmod 31$
   a. $\gamma = 17$, $\delta = 5$
   $t = \beta^\gamma \cdot \gamma^\delta = 6^{17} \cdot 17^5 \equiv 26 \cdot 26 \equiv 25 \bmod 31 \Rightarrow t = \alpha^x \Rightarrow ver(x,(\gamma,\delta)) = 1$
   (valid)
   b. $\gamma = 13$, $\delta = 15$
   $t = \beta^\gamma \cdot \gamma^\delta = 6^{13} \cdot 13^{15} \equiv 6 \cdot 30 \equiv 25 \bmod 31 \Rightarrow t = \alpha^x \Rightarrow ver(x,(\gamma,\delta)) = 1$
   (valid)

2. With p, α, d and thus β fixed, one can construct $\Phi(p-1)$ different ephemeral keys to compute the signature a message x. Here we have $\Phi(31-1) = \Phi(30) = 8$ possible k_E yielding 8 different signatures for a message x.

10.13

$$\begin{aligned} s_1 &\equiv (x_1 - d\,r)\,k_{E_1}^{-1} \bmod p-1 \\ s_2 &\equiv (x_2 - d\,r)\,k_{E_2}^{-1} = (x_2 - d\,r)\,(k_{E_1}+1)^{-1} \bmod p-1 \\ \Rightarrow \frac{s_1}{s_2} &\equiv \frac{(x_1 - d\,r)\,(k_{E_1}+1)}{(x_2 - d\,r)\,k_{E_1}} \bmod p-1 \\ \Leftrightarrow k_{E_1} &= \frac{1}{\frac{s_1\,(x_2 - d\,r)}{s_2\,(x_1 - d\,r)} - 1} \bmod p-1 \\ \Rightarrow d &\equiv \frac{x_1 - s_1\,k_{E_1}}{r} \bmod p-1 \end{aligned}$$

10.15 Similarly to the attack on Elgamal, an attacker can use following system of equations

$$\begin{aligned} s_1 &\equiv (SHA(x_1) + d\,r)\,k_E^{-1} \bmod q \\ s_2 &\equiv (SHA(x_2) + d\,r)\,k_E^{-1} \bmod q \end{aligned}$$

for known s_1, s_2, x_1, and x_2 to first compute the ephemeral key k_E and then the private key d:

$$\begin{aligned} s_1 - s_2 &\equiv k_E^{-1}(SHA(x_1) - SHA(x_2)) \bmod q \\ \Leftrightarrow k_E &\equiv \frac{SHA(x_1) - SHA(x_2)}{s_1 - s_2} \bmod q \\ \Rightarrow d &\equiv \frac{s_1 \cdot k_E - SHA(x_1)}{r} \bmod q \end{aligned}$$

10.17 If an attacker knows two consecutive signatures, he knows the values of $x_1, s_1, r_1, x_2, s_2, r_2$, and he knows the signature equations:

$$\begin{aligned} s_1 &= (x_1 - d \cdot r_1) \cdot k_E^{-1} \\ s_2 &= (x_2 - d \cdot r_2) \cdot (3 \cdot k_E)^{-1}. \end{aligned}$$

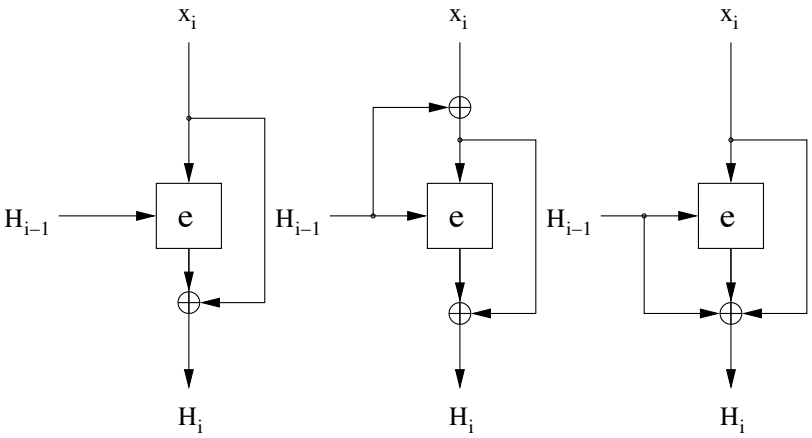
We now multiply both equations by the inverses, i.e.,

$$\begin{aligned} k_E \cdot s_1 &= (x_1 - d \cdot r_1) \\ 3 \cdot k_E \cdot s_2 &= (x_2 - d \cdot r_2) \end{aligned}$$

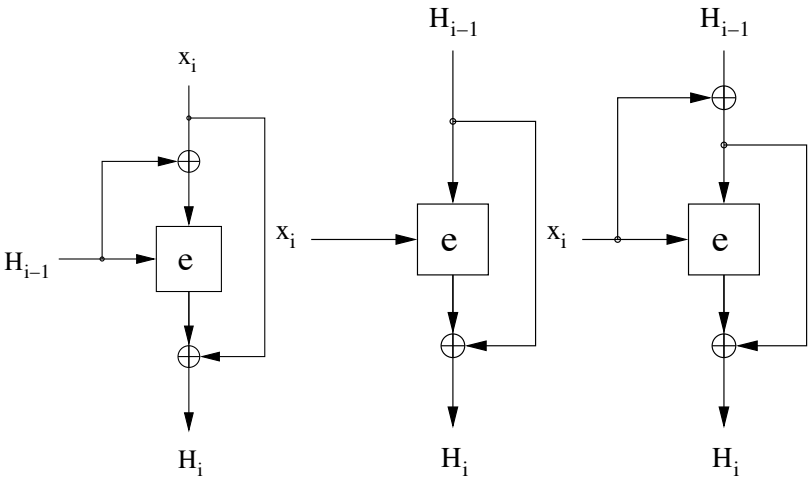
and obtain a system with two equations and two unknowns k_E and d which we can solve for d.

Problems of Chapter 11

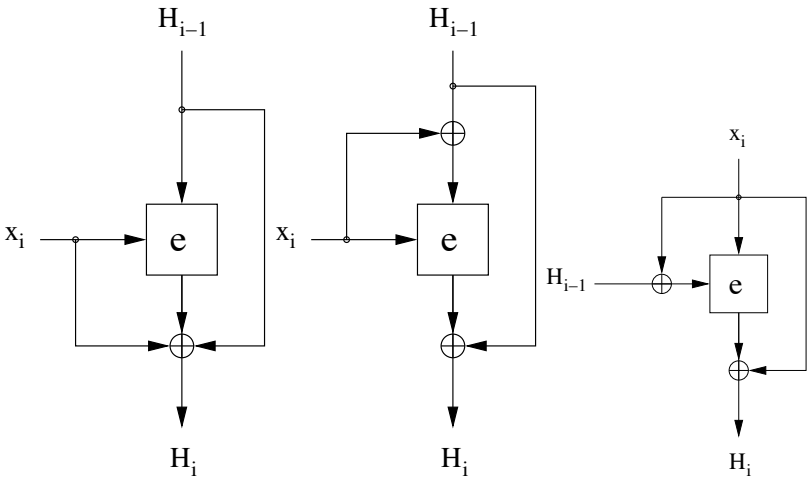
11.1



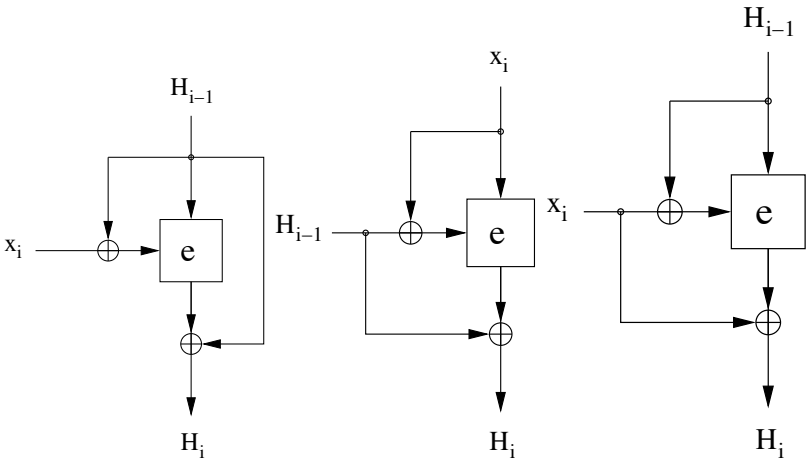
(a) $e(H_{i-1}, x_i) \oplus x_i$ (b) $e(H_{i-1}, x_i \oplus H_{i-1}) \oplus x_i \oplus H_{i-1}$ (c) $e(H_{i-1}, x_i) \oplus x_i \oplus H_{i-1}$



(d) $e(H_{i-1}, x_i \oplus H_{i-1}) \oplus x_i$ (e) $e(x_i, H_{i-1}) \oplus H_{i-1}$ (f) $e(x_i, x_i \oplus H_{i-1}) \oplus x_i \oplus H_{i-1}$



(g) $e(x_i, H_{i-1}) \oplus x_i \oplus H_{i-1}$ (h) $e(x_i, x_i \oplus H_{i-1}) \oplus H_{i-1}$ (i) $e(x_i \oplus H_{i-1}, x_i) \oplus x_i$



(j) $e(x_i \oplus H_{i-1}, H_{i-1}) \oplus H_{i-1}$ (k) $e(x_i \oplus H_{i-1}, x_i) \oplus H_{i-1}$ (l) $e(x_i \oplus H_{i-1}, H_{i-1}) \oplus x_i$

11.3

Birthday attack: $k \approx \sqrt{n \cdot m \cdot \frac{1}{1-\varepsilon}}$

n	$\varepsilon = 0.5$	$\varepsilon = 0.1$
2^{64}	$3.6 \cdot 10^9$	$1.4 \cdot 10^9$
2^{128}	$1.5 \cdot 10^{19}$	$6.0 \cdot 10^{18}$
2^{160}	$1.5 \cdot 10^{24}$	$3.9 \cdot 10^{23}$

number of messages after which probability for collision is ε

11.5

1. ■ Breaking the one-way property of a hash function would yield the desired passwords.
   ■ Finding second preimages works only when the attacker is already in possession of an existing password and hash value.
   ■ A collision of two (arbitrary) hash values (and passwords) will not yield an existing password.
2. Introducing a *salt* provides security against detecting similar passwords. I.e., if two or more passwords are the same, the hash values of such are similar, too. Hence, appending a (different) random value *salt* to each password prevents from a deterministic relationsship between passwords and their respective hashes. The above mentioned attacks are not affected by a salt.
3. Since the last property means a strong collision resistance is not required for this specific application, 80-bit hashes are sufficient.

11.7

1. 128 bits
2. If $c = 0$, both halfs of the output compute exactly the same 64 bits value. Hence, even though y_U has 128 bits, it only has an entropy of 64 bits. You, as an attacker, simply provide $(H_{0,L}, H_{0,R})$ and some start value for x_i (e..g., 64 zeros) as input to the hash function. You now search through possible passwords by incrementing x_i. This way, you will generate pseudo-random outputs y. Even though there is a chance you will not generate y_U at the output, the likelihood is small. Note that you can also try values x_i which have more than 64 bits by iterating the hash function.
3. A second-preimage attack
4. When $c \neq 0$ both halfs of the output will almost never be the same. So, the entropy of the output grows to (round about) 128 bits which makes a second-preimage attack computational infeasible.

11.9

1. $A_1 = T_1 + T_2 = H_0 + \Sigma_1^{\{256\}}(E_0) + Ch(E_0, F_0, G_0) + K_0 + W_0 + \Sigma_0^{\{256\}}(A_0) + Maj(A_0, B_0, C_0) = K_0 + W_0 = 428A2F98_{hex}$ $H_1 = G_0 = 00000000_{hex}$ $G_1 = F_0 = 00000000_{hex}$ $F_1 = E_0 = 00000000_{hex}$ $E_1 = D_0 + T_1 = D_0 + H_0 + \Sigma_1^{\{256\}}(E_0) + Ch(E_0, F_0, G_0) + K_0 + W_0 = K_1 + W_1 = 428A2F98_{hex}$ $D_1 = C_0 = 00000000_{hex}$ $C_1 = B_0 = 00000000_{hex}$ $B_1 = A_0 = 00000000_{hex}$
2. $A_1 = T_1 + T_2 = H_0 + \Sigma_1^{\{256\}}(E_0) + Ch(E_0, F_0, G_0) + K_0 + W_0 + \Sigma_0^{\{256\}}(A_0) + Maj(A_0, B_0, C_0) = K_0 + W_0 = 428A2F99_{hex}$ $H_1 = G_0 = 00000000_{hex}$ $G_1 = F_0 = 00000000_{hex}$ $F_1 = E_0 = 00000000_{hex}$ $E_1 = D_0 + T_1 = D_0 + H_0 + \Sigma_1^{\{256\}}(E_0) + Ch(E_0, F_0, G_0) + K_0 + W_0 = K_1 + W_1 = 428A2F99_{hex}$ $D_1 = C_0 = 00000000_{hex}$ $C_1 = B_0 = 00000000_{hex}$ $B_1 = A_0 = 00000000_{hex}$

11.11 Note that the input message is processed in chunks of r bits. From Table 11.3 we see that $r = 1344$ for 256 output bits, and for the 384 output bits variant it

holds that $r = 1088$. Thus, the latter implementation processes fewer input bits every time Keccak-f is computed. Hence, we need more evocations of Keccak-f in order to process the same number of message bits. As a result, the second implementation is by a factor of $1088/1344 \approx 0.81$ slower, resulting in a throughput of about 97 MBytes/s.

11.13

1. There are only two bit permutations. Either input bit 1 is connected to output bit 1 and input bit 2 to output bit 2, or the bits are crossed, i.e., input 1 is connected to output 2 and vice versa.
2. There are, of course, $2^2 = 4$ different inputs and outputs, namely $(00, 01, 10, 11)$. We can count the permutations as follows: The first input 00 can be mapped to 4 different output values. One has to pick one of those output values. The second input value can be mapped to 3 possible output values (one output value was previously assigned to the first input value). For the third input value, there are only 2 output values to choose from, and the fourth input value has to be assigned to the one remaining output value. Thus, in total there are

$$4 \cdot 3 \cdot 2 \cdot 1 = 4! = 24$$

   possible permutations. Here is the table:

Table 14.1 All permutations tables for a 2-bit permutation function

Input	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P16	P17	P18	P19	P20	P21	P22	P23	P24
00	00	00	00	00	00	00	01	01	01	01	01	01	10	10	10	10	10	10	11	11	11	11	11	11
01	01	01	10	10	11	11	00	00	10	10	11	11	00	00	01	01	11	11	00	00	01	01	10	10
10	10	11	01	11	01	10	10	11	00	11	00	10	01	11	00	11	00	01	01	10	00	10	00	01
11	11	10	11	01	10	01	11	10	11	00	10	00	11	01	11	00	01	00	10	01	10	00	01	00

3. Permutation P1 and P3 of f are identical to the two bit permuations.
4. There are $2^d!$ permutation functions and only $d!$ bit permutations

11.15

1. Of course, there are $5 \times 5 = 25$ bits with the value one. They form what is called the first slice of the state in the figure.
2. All bits have the value 0 except the 25 bits in the second slice, i.e., the bits $A[x, y, 1] = 1$.

Problems of Chapter 12

12.1

1. We use the Gaussian algorithm (mod 7) to solve the linear equation system.

$$\left(\begin{array}{ccc|c} 4&0&2&4\\ 0&5&4&5\\ 6&1&2&0 \end{array}\right) \to \left(\begin{array}{ccc|c} 1&0&4&1\\ 0&1&5&1\\ 6&1&2&0 \end{array}\right) \to \left(\begin{array}{ccc|c} 1&0&4&1\\ 0&1&5&1\\ 0&1&6&1 \end{array}\right) \to \left(\begin{array}{ccc|c} 1&0&4&1\\ 0&1&5&1\\ 0&0&1&0 \end{array}\right) \to \left(\begin{array}{ccc|c} 1&0&0&1\\ 0&1&0&1\\ 0&0&1&0 \end{array}\right)$$

   Therefore, $\mathbf{s} = \begin{pmatrix} 1\\1\\0 \end{pmatrix}$ is a solution.

3. If the error vector is known, the equation can be rearranged to

$$\begin{pmatrix} 4&0&2\\ 0&5&4\\ 6&1&2 \end{pmatrix} \cdot \mathbf{s} + \mathbf{e} = \begin{pmatrix} 6\\3\\2 \end{pmatrix} \Leftrightarrow \begin{pmatrix} 4&0&2\\ 0&5&4\\ 6&1&2 \end{pmatrix} \cdot \mathbf{s} = \begin{pmatrix} 6\\3\\2 \end{pmatrix} - \mathbf{e}$$
$$\Leftrightarrow \begin{pmatrix} 4&0&2\\ 0&5&4\\ 6&1&2 \end{pmatrix} \cdot \mathbf{s} = \begin{pmatrix} 6\\2\\2 \end{pmatrix}$$

   We solve this, as before, with a Gaussian elimination.

$$\left(\begin{array}{ccc|c} 4&0&2&6\\ 0&5&4&2\\ 6&1&2&2 \end{array}\right) \to \left(\begin{array}{ccc|c} 1&0&4&5\\ 0&1&5&6\\ 6&1&2&2 \end{array}\right) \to \left(\begin{array}{ccc|c} 1&0&4&5\\ 0&1&5&6\\ 0&1&6&0 \end{array}\right) \to \left(\begin{array}{ccc|c} 1&0&4&5\\ 0&1&5&6\\ 0&0&1&1 \end{array}\right) \to \left(\begin{array}{ccc|c} 1&0&0&1\\ 0&1&0&1\\ 0&0&1&1 \end{array}\right)$$

   Therefore, $\mathbf{s} = \begin{pmatrix} 1\\1\\1 \end{pmatrix}$ is a solution.

12.3

1. We calculate $d(x)$ as

$$\begin{aligned} d(x) &= a(x) \cdot b(x) + c(x) \\ &= (3x^3 + 4x^2 + x + 6)(4x^3 + 4x^2 + 5x + 6) + x^3 + 5x^2 + 3 \\ &= 12x^6 + 12x^5 + 15x^4 + 18x^3 + 16x^5 + 16x^4 + 20x^3 + 24x^2 \\ &\qquad + 4x^4 + 4x^3 + 5x^2 + 6x + 24x^3 + 24x^2 + 30x + 36 + x^3 + 5x^2 + 3 \\ &\equiv 5x^6 + 4x^3 + 2x^2 + x + 4 \\ &\equiv -5x^2 + 4x^3 + 2x^2 + x + 4 \\ &\equiv 4x^3 + 4x^2 + x + 4 \end{aligned}$$

2. We calculate $d(x)$ as

$$\begin{aligned}
d(x) &= a(x)\cdot b(x) + c(x) \\
&= (3x^3+4x^2+x+6)(4x^3+4x^2+5x+6) + x^3+5x^2+3 \\
&= 12x^6+12x^5+15x^4+18x^3+16x^5+16x^4+20x^3+24x^2 \\
&\qquad +4x^4+4x^3+5x^2+6x+24x^3+24x^2+30x+36+x^3+5x^2+3 \\
&\equiv x^6+6x^5+2x^4+x^3+3x^2+3x+6 \\
&\equiv -x^2-6x+-2+x^3+3x^2+3x+6 \\
&\equiv x^3+2x^2+8x+4
\end{aligned}$$

12.5

1.

$$\begin{aligned}
\mathbf{t}(x) &= (48x^3+16x^2+50x+51)(x^3+x^2+60)+x^2+60 \\
&= 48x^6+64x^5+66x^4+2981x^3+1012x^2+3000x+3120 \\
&\equiv 48x^6+3x^5+5x^4+53x^3+36x^2+11x+9 \\
&\equiv 53x^3+49x^2+8x+4
\end{aligned}$$

2. We encode $\bar{m} = 30x^2+30x+30$ and choose $r(x)=1, e_{aux}=60x, e_{msg}=x^2$ and calculate

$$\begin{aligned}
\mathbf{c}_{aux}(x) &= (48x^3+16x^2+50x+51)\cdot 1+60x \equiv 48x^3+16x^2+49x+51 \\
\mathbf{c}_{msg}(x) &= (53x^3+49x^2+8x+4)\cdot 1+x^2+30x^2+30x+30 = 53x^3+19x^2+38x+34
\end{aligned}$$

3. We compute

$$\begin{aligned}
\mathbf{m}'(x) &= 53x^3+19x^2+38x+34-(48x^3+16x^2+49x+51)(x^3+x^2+60) \\
&\equiv 13x^6+58x^5+57x^4+x^3+45x^2+26x+24 \\
&\equiv x^3+32x^2+29x+28
\end{aligned}$$

which we decode to $m=(0,1,1,1)$.

12.7 To determine the generator matrix G from the parity check matrix H, we construct a linear equation system as in Example 12.6. We know that the first four bits of our codewords consist of the message m and the last three bits belong to the redundancy r. Moreover, multiplying a valid codeword c with the parity check matrix H results in an all-zero vector.

$$H \cdot c^T = \begin{pmatrix} 1\,1\,1\,0\,1\,0\,0 \\ 1\,0\,1\,1\,0\,1\,0 \\ 1\,1\,0\,1\,0\,0\,1 \end{pmatrix} \begin{pmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ r_1 \\ r_2 \\ r_3 \end{pmatrix} \stackrel{!}{=} 0 \tag{14.7}$$

We can rewrite this equation to

$$\begin{array}{rrrrrrrrl} m_1 & + m_2 & + m_3 & & + r_1 & & & \equiv 0 & \mod 2 \\ m_1 & & + m_3 & + m_4 & & + r_2 & & \equiv 0 & \mod 2 \\ m_1 & + m_2 & & + m_4 & & & + r_3 & \equiv 0 & \mod 2 \end{array}$$

and reorganize the equations to

$$\begin{aligned} r_1 &\equiv m_1 + m_2 + m_3 \mod 2 \\ r_2 &\equiv m_1 + m_3 + m_4 \mod 2 \\ r_3 &\equiv m_1 + m_2 + m_4 \mod 2. \end{aligned}$$

Given these equations, we can easily construct our generator matrix G which is

$$G = \begin{pmatrix} 1\,0\,0\,0\,1\,1\,1 \\ 0\,1\,0\,0\,1\,0\,1 \\ 0\,0\,1\,0\,1\,1\,0 \\ 0\,0\,0\,1\,0\,1\,1 \end{pmatrix}.$$

12.9 To determine the messages m_1, m_2, m_3, and m_4, Bob first multiplies all four received codewords with the parity check matrix H to obtain the four syndromes s_1, s_2, s_3, and s_4.

$$s_1^T = H \cdot c_1'^T = \begin{pmatrix} 0\,1\,1\,1\,1\,0\,0 \\ 1\,1\,0\,1\,0\,1\,0 \\ 1\,0\,1\,1\,0\,0\,1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

$$s_2^T = H \cdot c_2'^T = \begin{pmatrix} 0\,1\,1\,1\,1\,0\,0 \\ 1\,1\,0\,1\,0\,1\,0 \\ 1\,0\,1\,1\,0\,0\,1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$$

$$s_3^T = H \cdot c_3'^T = \begin{pmatrix} 0\,1\,1\,1\,1\,0\,0 \\ 1\,1\,0\,1\,0\,1\,0 \\ 1\,0\,1\,1\,0\,0\,1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

$$s_4^T = H \cdot c_1'^T = \begin{pmatrix} 0\,1\,1\,1\,1\,0\,0 \\ 1\,1\,0\,1\,0\,1\,0 \\ 1\,0\,1\,1\,0\,0\,1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Given the four syndromes, Bob detects that the codewords c_2' and c_3' contain errors that need to be corrected. Using the look-up table from Example 12.7, he can easily determine the error vectors, e.g.,

$$e_2 = \begin{pmatrix} 0\,0\,1\,0\,0\,0\,0 \end{pmatrix}$$

and

$$e_3 = \begin{pmatrix} 0\,0\,0\,1\,0\,0\,0 \end{pmatrix}.$$

To this end, the corrected codewords are

$$c_1 = \begin{pmatrix} 1\,0\,0\,0\,0\,1\,1 \end{pmatrix} \qquad c_2 = \begin{pmatrix} 1\,0\,0\,1\,1\,0\,0 \end{pmatrix}$$
$$c_3 = \begin{pmatrix} 1\,0\,0\,1\,1\,0\,0 \end{pmatrix} \qquad c_4 = \begin{pmatrix} 1\,0\,0\,1\,1\,0\,0 \end{pmatrix}$$

which results in the four messages

$$m_1 = \begin{pmatrix} 1\,0\,0\,0 \end{pmatrix} \qquad m_2 = \begin{pmatrix} 1\,0\,0\,1 \end{pmatrix}$$
$$m_3 = \begin{pmatrix} 1\,0\,0\,1 \end{pmatrix} \qquad m_4 = \begin{pmatrix} 1\,0\,0\,1 \end{pmatrix}.$$

The corresponding characters for $(m_1|m_2)$ and $(m_3|m_4)$ are E and U, respectively.

12.11 First, compute the intermediate value u by

$$u = c \cdot P^{-1} = \begin{pmatrix} 1\,1\,1\,0\,1\,0\,1 \end{pmatrix} \cdot \begin{pmatrix} 1\,0\,0\,0\,0\,0\,0 \\ 0\,0\,0\,1\,0\,0\,0 \\ 0\,0\,0\,0\,0\,1\,0 \\ 0\,1\,0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0\,0\,1 \\ 0\,0\,0\,0\,1\,0\,0 \\ 0\,0\,1\,0\,0\,0\,0 \end{pmatrix} = \begin{pmatrix} 1\,0\,1\,1\,0\,1\,1 \end{pmatrix}.$$

Second, we compute the syndrome s' by multiplying H with u resulting in

$$s' = H \cdot u^T = \begin{pmatrix} 1\,0\,1\,1\,1\,0\,0 \\ 1\,1\,0\,1\,0\,1\,0 \\ 1\,1\,1\,0\,0\,0\,1 \end{pmatrix} \cdot \begin{pmatrix} 1\,0\,1\,1\,0\,1\,1 \end{pmatrix}^T = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

Third, given the syndrome s' and the parity check matrix H, we can determine the permuted error vector e' by identifying the column of H that matches s'. In our case, this is the first column which leaves us with $e' = \begin{pmatrix} 1\,0\,0\,0\,0\,0\,0 \end{pmatrix}$. Fourth, we remove the error from u to get an error-free vector $\tilde{u}$:

$$\tilde{u} = u + e' = \begin{pmatrix} 0\,0\,1\,1\,0\,1\,1 \end{pmatrix}$$

Fifth, we can extract the first four bits to obtain $v = \begin{pmatrix} 0\,0\,1\,1 \end{pmatrix}$. Sixth, we compute our message m by

$$m = v \cdot S^{-1} = \begin{pmatrix} 0\,0\,1\,1 \end{pmatrix} \cdot \begin{pmatrix} 1\,1\,1\,1 \\ 0\,1\,1\,0 \\ 0\,0\,0\,1 \\ 1\,0\,1\,1 \end{pmatrix} = \begin{pmatrix} 1\,0\,1\,0 \end{pmatrix}$$

which is the message generated by Alice.

12.13

1. The message length is $8 * 1000^2$ bits. Thus, the length of the public key calculates as $2n^2 = 2 * (8 * 1000^2)^2 = 128\ TB$.
2. SHA-256 has an output length of 256 bits. Hence, the public key length for the hashed message is $2n^2 = 2 * 256^2\ bits = 131072\ bits$.
3. A new key is required for every signature generation. Thus, the total size will be $356 * 131072\ bits = 46661632\ bits \approx 5.8\ MB$.

12.15 To sign a message of length $n = 9\ Bits$, we need a checksum that can handle values from 0 to $\lceil l/w \rceil \cdot (2^w - 1) = 21$. Thus, we need we need $l = \lceil n/w \rceil = 3$ blocks for the message and $\lceil \lceil log_2(21) \rceil / w \rceil = 2$ blocks for the checksum. Thus, the private key is $k_{pr} = (184, 245, 20, 60, 311)$. To generate the public key, we need to apply $f(x)$, $2^w - 1$ times to each private key block:

Step	Chain 0	Chain 1	Chain 2	Chain 3	Chain 4
0	184	245	20	**60**	311
1	130	238	400	23	142
2	37	434	57	18	235
3	347	308	183	324	37
4	324	329	**274**	221	347
5	**221**	420	470	296	324
6	296	**105**	148	235	**221**
7	235	294	442	37	296

Hence, the public key is $k_{pub} = (235, 294, 442, 37, 296)$. To generate the signature, we need to split m into base-w blocks: $m = 101|110|100 = 5|6|4$. Then, we need to compute the checksum $c = 7-5+7-6+7-4 = 6$ and convert it to base-w representation $c = 000|110 = 0|6$. From here, we can use our calculations from above to read the signature $sig = (221, 105, 274, 60, 221)$. The validity of the signature follow directly from the table, since completing the hash chain for each block will result in the public key.

12.17 For the verification, the receiver needs to perform the following operations:

- Complete the hash-chains of the W-OTS signature to receive Y'_0.
- Compress Y'_0 using $h(x)$ to obtain $v_0[0]'$.
- Use the authentication path to compute the root node of the tree: $v_1[0]' = h(v_0[0]', v_0[1])$, $v_2[0]' = h(v_1[0]', v_1[1])$, $v_3[0]' = h(v_2[0]', v_2[1])$, $k'_{pub} = h(v_3[0]', v_3[1])$.
- Finally, the receiver must compare k'_{pub} and k_{pub}. Only if these values are equal, the signature is valid.

For Y_8, the authentication path is $(v_0[9], v_1[5], v_2[3], v_3[0])$. For the verification, the receiver needs to perform the following operations:

- Complete the hash-chains of the W-OTS signature to receive Y'_8.
- Compress Y'_8 using $h(x)$ to obtain $v_0[8]'$.
- Use the authentication path to compute the root node of the tree: $v_1[4]' = h(v_0[8]', v_0[9])$, $v_2[2]' = h(v_1[4]', v_1[5])$, $v_3[1]' = h(v_2[2]', v_2[3])$, $k'_{pub} = h(v_3[0], v_3[1]')$.
- Finally, the receiver must compare k'_{pub} and k_{pub}. Only if these values are equal, the signature is valid.

12.19 First, we need to generate the binary tree. Therefore, we need to compute the intermediate nodes of the hash tree:

1. *Tree Level 1:*
   - *Leaf Nodes 0 and 1:* $v_1[0] = f(411+245) = 240$
   - *Leaf Nodes 2 and 3:* $v_1[1] = f(44+192) = 4$
   - *Leaf Nodes 4 and 5:* $v_1[2] = f(376+199) = 314$
   - *Leaf Nodes 6 and 7:* $v_1[3] = f(418+53) = 370$
2. *Tree Level 2:*
   - *Nodes 0 and 1:* $v_2[0] = f(240+4) = 316$
   - *Nodes 2 and 3:* $v_2[1] = f(314+370) = 68$
3. *Tree Level 3:*
   - *Nodes 0 and 1:* $v_3[0] = f(316+68) = 128$

The final signature is $sig = (s = 4, sig_{OTS} = (72, 300, 220, 436, 52), auth\_path = (44, 240, 68))$. It is important to update the state.

Problems of Chapter 13

13.1

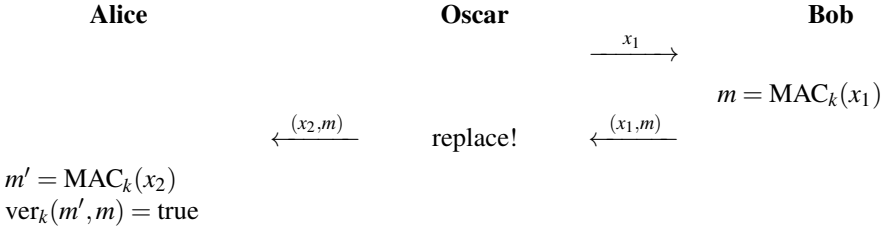
1. ■ Calculate $x||h = e_{k_1}^{-1}(y)$.
   ■ Calculate $h' = H(k_2||x)$.
   ■ If $h = h'$, the message is authentic. If $h \neq h'$, either the message or the MAC (or both) has been altered during transfer.
2. ■ Calculate $x||s = e_{k_1}^{-1}(y)$.
   ■ Calculate $h' = H(x)$.
   ■ Verify the signature: $ver_{k_{pub}}(s, H(x))$

13.3

1. $c_i = z_i \bigoplus \{x_1 x_2 \ldots x_n || H_1(x) H_2(x) \ldots H_m(x)\}; \qquad i = 1, 2, \ldots, n+m$
   where $H_j(x)$ is the j^{th} bit of the m-bit hash value $h(x)$.
   1) Assume x has n bits. With the knowledge of x, Oscar first computes
   $z_i = x_i \oplus c_i; \qquad i = 1, 2, \ldots, n$
   2) Oscar recomputes $H(x)$ since he knows x.
   3) Assume $H(x)$ has m output bits. Oscar computes
   $z_{j+n} = H_j(x) \oplus c_{j+n}; \qquad j = 1, 2, \ldots, m$
   4) For a different x', Oscar computes $H(x')$
   5) Oscar then computes
   $c'_i = z_i \oplus x'_i; \qquad i = 1, 2, \ldots, n$
   $c'_{j+n} = z_{j+n} \oplus H_j(x'); \qquad j = 1, 2, \ldots, m$
2. No. Although Oscar can still recover $z_1, z_2, \ldots, z_n$, he cannot recover the bitstream portion $z_{n+1}, z_{n+2}, \ldots, z_{n+m}$ which was used for encrypting $\text{MAC}_{k_2}(x)$. Even if he would know the whole bitstream, he would not be able to compute a valid $\text{MAC}_{k_2}(x')$ since he does not know k_2.

13.5

1. This attack assumes that Oscar can trick Bob into signing the message x_1. This is, of course, not possible in every situation, but one can imagine scenarios where Oscar can pose as an innocent party and x_1 is the message being generated by Oscar.



2. For constructing collisions, Oscar must be able to compute about $\sqrt{2^n}$ MACs, where n is the output width. Since Oscar does not have the secret key, he has to

somehow trick Alice and/or Bob into computing MACs for that many messages, as shown above. Even though it might work for a few messages (cf. above), it can be very difficult to do it for the large number of messages needed for collision finding. On the other hand, collisions for hash functions can be constructed by Oscar himself off-line, with massive computing resources if needed, without the help of Alice and Bob because these computations are un-keyed.
A 128-bit MAC provides, thus, a security of 2^{128} since collision attacks are not applicable. A hash function with the same output size offers only a security of about 2^{64}.

13.7

1. $\text{MAC}_k(x) = H(k||x)$

$$x = (x_1, ..., x_n)$$
$$m = h(k||x_1, ..., x_n)$$

Modification of an attacker:

$$x_0 = (x_1, ..., x_n, x_{n+1})$$
$$m_0 = h(m||x_{n+1})$$

Recipient:

$$m' = h(k||x_1, ..., x_n, x_{n+1})$$
$$m' = m_0$$

2. $\text{MAC}_k(x) = H(x||k)$ The attcker needs to find a collision for the message in order to compute a valid MAC:

$$h(x) = h(x_0)$$
$$m = h(x||k) = h(x_0||k)$$

Problems of Chapter 14

14.1

1. (1) Session keys are derived by a linear and invertible(!) operation of the previous session key.
   (2) Usage of hash functions, thus a non-linear correlation of the session keys.
   (3) Usage of the masterkey and the previous session key for every derivation of the next session key.
2. Methods (2) and (3), since the old session keys cannot be extracted from the recent session key

3. (1) every session, since PFS is missing
   (2) every session using the hacked session key K_n and every following session
   (3) only the recent session, since the (unknown) masterkey is used for every furterh key derivation
4. No, since then, all session keys can be calculated!

14.3

The first class of encryptions (between the KDC and a user) should be done using AES-256. The session between two arbitrary users (the second class) should be encrypted using PRESENT. Here is the rationale for it:

PRESENT with an 80-bit key does not provide good long-term security, even though it is currently secure against brute-force attacks, at least considering non-governmental adversaries. Should an attacker be able to compute a session key via brute-force attack, which is most likely be possible once quantum computers become available in the future, only the corresponding session will be compromised. On the other hand, AES-256 appears to provide excellent long-term security (even against quantum computers) and should be used for as the long-term keys between the KDC and each user. We need much stronger security here because if it becomes possible to find a certain $K_{U,KDC}$, all previous and future communication of the user U could be eavesdropped.

14.5

Assuming that the hacker obtains the key $K^i_{U,KDC}$, he can initially encrypt recent session data in which the session keys K_{ses} are encrypted with $K^i_{U,KDC}$. He will also be able to decrypt the subsequent keys $K^{i+j}_{U,KDC}$ until the attack is detected at time t_y. At this point, new keys are exchanged using a secure channel. Hence, all communication between t_x and t_y may be compromised. Crucially, the attacker is, even with knowledge of $K^i_{U,KDC}$, not able to recover $K^{i-1}_{U,KDC}$. Hence, he cannot decrypt messages prior to time t_x. In conclusion, this variant provides Perfect Forward Secrecy.

14.7

1. Once Alice's KEK k_A is being compromised, Oscar can compute the session key k_{ses} and, thus, decrypt all messages.
2. The same applies to a compromised KEK k_B of Bob.

14.9

1. $t = 10^6$bits/s
   storage $= t \cdot r = 2h \cdot 10^6$ bits/s $= 2 \cdot 3600 \cdot 10^6$ bits/s $= 7.2$ Gbits $= 0.9$ GB
   Soring several Gbyte can be easily done, e.g., on cheap USB sticks..
2. We compute the number of keys that an attacker can recover in 30 days:

$$\#\,\text{Keys} = \frac{30\,days}{10\,min} = \frac{30 \cdot 24 \cdot 60}{10} = 4320$$

   Key derivation period:

$$T_{Kdev} = \frac{2h}{4320} = 1.67\ \text{sec}$$

Since hash functions are fast, a key derivation can easily be performed (in software) at such a rate.

14.11

- Alice computes:
  $A = 2^{228} \equiv 394 \bmod 467$
  $k_{AO} = O^a = 156^{228} \equiv 243 \bmod 467$
- Bob computes:
  $B = 2^{57} \equiv 313 \bmod 467$
  $k_{BO} = O^b = 156^{57} \equiv 438 \bmod 467$
- Oscar computes:
  $O = 2^{16} \equiv 156 \bmod 467$
  $k_{AO} = A^o = 394^{16} \equiv 243 \bmod 467$, $k_{BO} = B^o = 313^{16} \equiv 438 \bmod 467$,

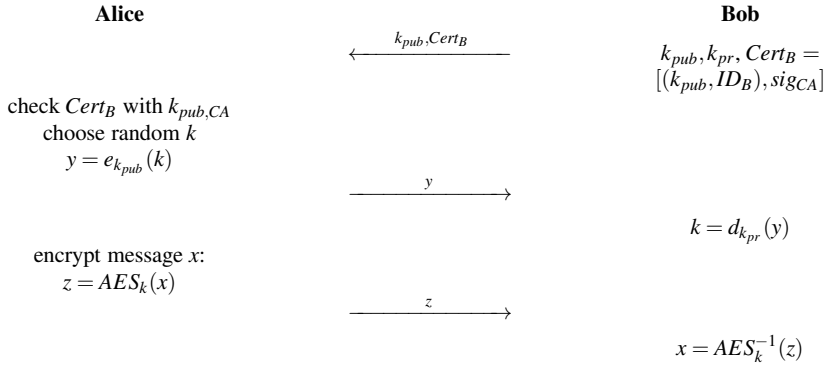
14.13

1. Alice would detect this forgery because the certificate $C(O)$ will contain the ID of Oscar and not ID(B). She will, thus, know immediately that this is not Bob's certificate.
2. This kind of forgery would be detected by Alice when she verifies the certificate with the CA's public key. Since the payload of the certificate (namely the ID) has been altered, the signature verification will naturally fail.

14.15

The signature of the CA merely covers the *public* key of a user U, i.e., $k_{pub_U} = \alpha^{a_U}$. Even if Oscar gets access to all the private key of the CA's signature algorithm, he still cannot compute the private key of any user (which would require to solve the discrete logarithm problem). Thus, Oscar can also not calculate the symmetric session keys, which were used before he obtained the CA's signature key. However, from now on he is capable of masquerading as any user and can generate fake certificates for any user.

14.17



14.19 PGP makes use of the so called *Web of Trust* architecture. In contrast of tree-based architectures, a WoT only consists of nodes (users) with equal rights, where each node can certify other nodes. The validity of a certificate is then given through a *Chain of Trust*. In principle, Alice trusts certificates that she received from a user she know, e.g., from Bob. (Ideally, Alice would in fact verify the certificate, e.g., by checking the certificate's fingerprint with Bob.) In a Chain of Trust, she also trusts in the certificates that Bob trusts and so on. The main advantage of this system is the lack of mutually trusted CA. The drawback, however, is that it must be assured that all users are honest and don't accept fake certificates.

14.21

1. ■ Bob sends a nonce n_B to Alice (which will be used later in the protocol to prevent replay attacks).
   ■ Alice signs Bob's value n_B together with her own nonce n_A and with Bob's identity ID_B. She sends this signature together with n_A as "message" to Bob.
   ■ Now, Bob verifies the signature with Alice's public key and learns the following if the verification is correct. He knows that the message is not a replayed one since Alice has signed the current nonce n_B. Moreover, he knows that Alice is actually authenticating herself towards Bob since ID_B was signed.
   To confirm to Alice that he has in fact received the information correctly, he sends a signature over n_A to Alice together with a newly generated random value n'_B and Alice's identity ID_A.
   ■ Alice can now verify the message with Bob's public key. From that she learns that the message is actually coming from Bob if the signature checks out. She also knows that it is not the replay of an older message because it contains her current nonce n_A. Moreover, she knows that Bob wants to authenticate himself towards Alice since ID_A was signed.
2. The core idea of the attack is that Oscar only forwards the messages he receives from the two parties. The consequences are interesting: Oscar impersonates Alice when communicating with Bob, i.e., Bob assumes that Oscar in fact is Alice. Oscar also impersonates Bob towards Alice, i.e., she thinks that the person she's communicating with actually is Bob.
   Interleaving attacks are very difficult to prevent because there Oscar does not manipulate any cryptographic secrets (keys) but merely is in the middle of a legitimate message exchange between Alice and Bob.
3. The interleaving attack is targeting *authentication* protocols and leads, thus, to a violation of (entity) authentication. In contrast, the MITM attack is an attack against a key exchange and leads to incorrect establishment of keys between users. The consequences of the MITM attack are, thus, much more far reaching since incorrect keys can be used by the adversary to break many security services, including confidentiality and integrity.