

Answers to the Exercises

2. Symmetric-Key Encryption

1. If all keys are equal, then $C_0 = 0 \dots 0$ or $C_0 = 1 \dots 1$.

We consider for example the bits at the positions

2,3,5,7,9,11,13,15,16,18,20,22,24,26,28,1 of C_0 and denote this sequence by b_1, b_2, \dots, b_{16} .

Bit b_i appears as bit number 5 in k_i , $i = 1, \dots, 16$. Thus we have $b_1 = b_2 = \dots = b_{16}$, because all keys are equal. Additionally we consider the positions 3,4,6,8,10,12,14,16,17,19,21,23,25,27,1,2 of C_0 . The i -th bit in this sequence is the 24th bit of k_i . Thus all bits at these positions are equal. Position 3 appears in both cases. Thus all bits of C_0 are equal.

Similar arguments show that $D_0 = 0 \dots 0$ or $D_0 = 1 \dots 1$.

We obtain the four weak keys by combining the possible values of C_0 and D_0 . If we apply *PC1* to the four rows

```

01 01 01 01 01 01 01 01
FE FE FE FE FE FE FE FE
1F 1F 1F 1F 0E 0E 0E 0E
E0 E0 E0 E0 F1 F1 F1 F1

```

we see that the four rows are the weak keys of DES. Note that *PC1* is a permutation on 56 bits. The bits in the positions 8,16,24,32,40,48,56,64 are not used.

2. a. Note that \bar{k} yields $\overline{k_i}$, if k yields k_i and that $E(\bar{x}) = \overline{E(x)}$. Thus

$$f(\bar{x}, \bar{k}) = P(S(E(\bar{x}) \oplus \bar{k})) = P(S(E(x) \oplus k)) = f(x, k)$$

and

$$\begin{aligned}
 \phi_i(\bar{x}, \bar{y}) &= (\bar{x} \oplus f_i(\bar{y}), \bar{y}) \\
 &= (\bar{x} \oplus f(\bar{y}, \bar{k}_i), \bar{y}) \\
 &= (\bar{x} \oplus f(y, k_i), \bar{y}) \\
 &= (\bar{x} \oplus f_i(y), \bar{y}) \\
 &= \overline{(x \oplus f_i(y), y)} \\
 &= \overline{\phi_i(x, y)}.
 \end{aligned}$$

Hence we get

$$\begin{aligned} DES_{\bar{k}}(\bar{x}) &= IP^{-1}(\phi_{16}(\mu(\phi_{15}(\dots\mu(\phi_2(\mu(\phi_1(IP(\bar{x}))))))\dots)))) \\ &= IP^{-1}(\phi_{16}(\mu(\phi_{15}(\dots\mu(\phi_2(\mu(\phi_1(\overline{IP(x)}))))))\dots)))) \\ &\vdots \\ &= \overline{IP^{-1}(\phi_{16}(\mu(\phi_{15}(\dots\mu(\phi_2(\mu(\phi_1(IP(x))))))\dots))))} \\ &= \overline{DES_k(x)}. \end{aligned}$$

b. $DES(k, \bar{x}) = y$ implies

$$DES(\bar{k}, x) = DES(\bar{k}, \bar{x}) = \overline{DES(k, \bar{x})} = \bar{y}.$$

Assume $c = DES_k(m)$ and $\tilde{c} = DES_k(\tilde{m})$ are known.

Choose k' and compute $y = DES(k', \bar{m})$.

i. If $y = \tilde{c}$, then the key is k' .

ii. If $\bar{y} = c$, then the key is \bar{k}' .

Thus, we can test the two keys k' and \bar{k}' with one encryption.

3. Let $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$ be a permutation, x_1 an initial value and x_1, x_2, \dots the sequence obtained by applying f . Then there exists an i with $f(x_{i+1}) \in \{x_1, \dots, x_i\}$. Let j be the first i with this property. Since f is a permutation $f(x_i) = x_1$. Otherwise an element would have two pre-images. (x_1, \dots, x_j) is a cycle of f . The average period of the key stream is the average length of a cycle of a randomly selected permutation.

Let $S = \{0, \dots, k\}$ and

$$C_m = \{c \mid c \text{ is a cycle of length } m \text{ of a permutation on } S\}.$$

A fixed cycle of length m appears in $(n-m)!$ permutations. The number of different cycles (x_1, \dots, x_m) is $\frac{k(k-1)\dots(k-m+1)}{m}$. Thus

$$|C_m| = \frac{k!}{m}.$$

Let $C_{m,l} = \{c \in C_m \mid c \text{ contains } l\}$. Totally there appear $k!$ elements in cycles of length m . Each element l is equally likely to appear. Thus

$$|C_{m,l}| = \frac{k!}{k}$$

(independent of m and l). The average number of cycles of length m containing l over all permutations is $\frac{1}{k}$. We get as average over all cycle-lengths

$$\sum_{m=1}^k \frac{m}{k} = \frac{k+1}{2}.$$

For $n = 2^k$ we get an average cycle-length of $2^{n-1} + \frac{1}{2}$.

3. Public-Key Cryptography

2. By the Chinese remainder theorem we have

$$\mathbb{Z}_n^* \cong \mathbb{Z}_p^* \times \mathbb{Z}_q^*$$

and μ decomposes into

$$(\mu_1, \mu_2) : \mathbb{Z}_p^* \times \mathbb{Z}_q^* \longrightarrow \mathbb{Z}_p^* \times \mathbb{Z}_q^*, (x_1, x_2) \longmapsto (x_1^e, x_2^e)$$

μ is an isomorphism if and only if μ_1, μ_2 are isomorphisms. Now

$$\mu_1 : \mathbb{Z}_p^* \longrightarrow \mathbb{Z}_p^*, x \longmapsto x^e$$

and

$$\mu_2 : \mathbb{Z}_q^* \longrightarrow \mathbb{Z}_q^*, x \longmapsto x^e$$

are isomorphisms if and only if $\gcd(e, p-1) = 1$ and $\gcd(e, q-1) = 1$. This implies the assertion.

3. Let g be a primitive root in \mathbb{Z}_p^* .

$$\text{Exp} : \mathbb{Z}_{p-1} \longrightarrow \mathbb{Z}_p^*, \alpha \longmapsto g^\alpha$$

is an isomorphism of groups. Let $k \in \mathbb{N}, x \in \mathbb{Z}_p^*$ and $x^k = 1$. Then $x = g^\nu$ and $x^k = g^{\nu k}$. Hence $p-1$ divides νk . This implies

$$\begin{aligned} |\{x \in \mathbb{Z}_p^* \mid x^k = 1\}| &= |\{\nu \in \mathbb{Z}_{p-1} \mid \nu k \equiv 0 \pmod{p-1}\}| \\ &= |\{\frac{p-1}{d}l \mid 1 \leq l \leq d\}| = d, \end{aligned}$$

where $d = \gcd(k, p-1)$.

Now $\mathbb{Z}_n^* \cong \mathbb{Z}_p^* \times \mathbb{Z}_q^*$ and $x^{e-1} = 1$ if and only if $(x_1^{e-1}, x_2^{e-2}) = (1, 1)$, where $x_1 = x \pmod{p}$ and $x_2 = x \pmod{q}$. This implies

$$|\{x \in \mathbb{Z}_n^* \mid \text{RSA}_e(x) = x\}| = \gcd(e-1, p-1) \gcd(e-1, q-1).$$

4. Compute $\lambda = ed - 1, \lambda = 2^t m, m$ odd. λ is a multiple of $\varphi(n)$. Thus $[a^\lambda] = 1$ for all $[a] \in \mathbb{Z}_n^*$. Let

$$W_n := \left\{ [a] \in \mathbb{Z}_n^* \mid a^m \equiv 1 \pmod{n} \right. \\ \left. \text{or there is an } i, 0 \leq i \leq t-1, \text{ with } a^{2^i m} \equiv -1 \pmod{n} \right\}.$$

Let $[a] \notin W_n$. Then there is an $i, 0 \leq i \leq t-1$, with $a^{2^{i+1}m} \equiv 1 \pmod{n}$ and $a^{2^i m} \not\equiv \pm 1 \pmod{n}$. Then $[a^{2^i m}]$ and $[1]$ are square roots of $[1]$, and the

factors of n can be computed by the Euclidean algorithm (Lemma A.63). Let $\overline{W}_n := \mathbb{Z}_n^* \setminus W_n$ be the complement of W_n . Then $|\overline{W}_n| \geq \frac{\varphi(n)}{2}$ (see below). Hence, choosing a random $[a] \in \mathbb{Z}_n^*$ we can compute the factors of n in this way with probability $\geq 1/2$, since $[a]$ is not in W_n with a probability $\geq 1/2$. Repeating the random choice t -times, if necessary, we can increase the probability of success to $\geq 1 - 2^{-t}$. It remains to show that $|\overline{W}_n| \geq \frac{\varphi(n)}{2}$.

$$W_n^i := \{[a] \in \mathbb{Z}_n^* \mid a^{2^i m} \equiv -1 \pmod{n}\}.$$

W_n^0 is not empty, since $[-1] \in W_n^0$. Let $r = \max\{i \mid W_n^i \neq \emptyset\}$ and

$$U := \{a \in \mathbb{Z}_n^* \mid a^{2^r m} \equiv \pm 1 \pmod{n}\}.$$

U is a subgroup of \mathbb{Z}_n^* and $W_n \subset U$.

Let $[x] \in W_n^r$. By the Chinese Remainder Theorem A.29, there is a $[w] \in \mathbb{Z}_n^*$ with $w \equiv x \pmod{p}$ and $w \equiv 1 \pmod{q}$. Then $w^{2^r m} \equiv -1 \pmod{p}$ and $w^{2^r m} \equiv +1 \pmod{q}$, hence $w^{2^r m} \not\equiv \pm 1 \pmod{n}$. Thus, $w \notin U$, and we see that U is indeed a proper subgroup of \mathbb{Z}_n^* . Thus $|W_n| \leq \frac{\varphi(n)}{2}$.

6. a. Let $R_{p'} := \{x \in \mathbb{Z}_p^* \mid p' \text{ does not divide } \text{ord}(x)\}$.

Note that

- i. p' does not divide $\text{ord}(x)$, if and only if $\text{ord}(x)$ divides a , and that
- ii. p' divides ν , where ν is defined by a representation $x = g^\nu$ of x with a primitive root g .

Thus,

$$|R_{p'}| = |\{x \in \mathbb{Z}_p^* \mid \text{ord}(x) \mid a\}| = |\{g^{p^l} \mid 1 \leq l \leq a\}| = a.$$

- b. Let $R_{p'q'} := \{x \in \mathbb{Z}_n^* \mid p'q' \text{ does not divide } \text{ord}(x)\}$.

Note $\text{ord}(x)$ is the least common multiple of $\text{ord}(x \pmod{p})$ and $\text{ord}(x \pmod{q})$. $p'q'$ does not divide $\text{ord}(x)$, if and only if p' does not divide $\text{ord}(x \pmod{p})$ or q' does not divide $\text{ord}(x \pmod{q})$. By the Chinese Remainder Theorem, we have

$$\begin{aligned} \mathbb{Z}_p^* \times R_{q'} \cup R_{p'} \times \mathbb{Z}_q^* &= R_{p'q'}, \\ \mathbb{Z}_p^* \times R_{q'} \cap R_{p'} \times \mathbb{Z}_q^* &= R_{p'} \times R_{q'}. \end{aligned}$$

This implies $|R_{p'q'}| = (p-1)b + a(q-1) - ab$ and

$$\frac{|R_{p'q'}|}{\varphi(n)} = \frac{(p-1)b + a(q-1) - ab}{ap'bq'} = \frac{1}{p'} + \frac{1}{q'} - \frac{1}{p'q'}.$$

7. a. $\text{RSA}_e^l(x) = x^{e^l} = x$ if $e^l \equiv 1 \pmod{\varphi(n)}$. The last condition is satisfied for $l = \varphi(\varphi(n))$.

- b. $x^{e^i} = x$ is equivalent to $x^{e^i - 1} = 1$. The last equation is equivalent to $e^i \equiv 1 \pmod{\text{ord}(x)}$.

To prevent the decryption-by-iterated-encryption attack, it is required that $\text{ord}(e \pmod{\text{ord}(x)})$ is large for x and e .

We show that the set of “exceptions”,

$$\{(x, e) \in \mathbb{Z}_n^* \times \mathbb{Z}_{\varphi(n)}^* \mid \text{ord}(e \pmod{\text{ord}(x)}) < p''q''\},$$

is an exponentially small subset of $\mathbb{Z}_n^* \times \mathbb{Z}_{\varphi(n)}^*$. The frequency of elements $x \in R_{p'q'}$ (see Exercise 6) is exponentially small. Let $x \notin R_{p'q'}$. Then $n' = p'q'$ divides k , $k := \text{ord}(x)$. Then $\text{ord}(e \pmod{n'})$ divides $\text{ord}(e \pmod{k})$.

Thus, if $p''q''$ divides $\text{ord}(e \pmod{n'})$ then $p''q''$ divides $\text{ord}(e \pmod{k})$ and $\text{ord}(e \pmod{k})$ is large.

Let $R_{p''q''} := \{e \in \mathbb{Z}_n^* \mid p''q'' \text{ does not divide } \text{ord}(e)\}$.

Let f be the frequency of elements $e \in R_{p''q''}$.

By Exercise 6, $f = \frac{1}{p''} + \frac{1}{q''} - \frac{1}{p''q''}$ is exponentially small. The discussion shows that $p''q''$ is a lower bound for the number of iterations of the repeat-until loop for all (x, e) outside an exponentially small subset of $\mathbb{Z}_n^* \times \mathbb{Z}_{\varphi(n)}^*$.

8. Elements (x, y) in the domain of f are bit-strings of length $2(|q - 1|)$. Elements in the range $G_q \subset \mathbb{Z}_p^*$ are encoded as bit strings of length $|p|$. Since $|q - 1| = |q| = |p| - 1$, we may consider f as a compression function. Assume $(x_1, y_1), (x_2, y_2)$ is a collision of f . Then $g^{x_1}h^{y_1} = g^{x_2}h^{y_2}$. Thus $g^{x_1 - x_2} = h^{y_2 - y_1}$. If $y_1 = y_2$, then $x_1 = x_2$ and $(x_1, y_1) = (x_2, y_2)$. This is a contradiction, since $(x_1, y_1), (x_2, y_2)$ cannot be equal, as a collision of f . Thus $y_1 \neq y_2$. We get $\log_g h = \frac{x_1 - x_2}{y_2 - y_1}$.
9. Given a value $v \in \{0, 1\}^n$, we randomly select messages $m \in \{0, 1\}^*$ and check, if $h(m) = v$. We don't have to check, if we have selected a message twice. The probability for this event is negligibly small (note that the set of messages is infinite). The probability that $h(m) = v$ is $1/2^n$. Lemma B.12 – with $p = 1/2^n$ – says that we expect to select 2^n messages m until $h(m) = v$. The expected number of steps does not depend on v , and we conclude immediately that the expected number of steps in the brute-force attack against the one-way property of h is 2^n . To attack second pre-image resistance, we consider a message $m' \in \{0, 1\}^*$. Let $v = h(m')$. We randomly select messages $m \in \{0, 1\}^*$, $m \neq m'$ and check, if $h(m) = v$. As before, the probability that $h(m) = v$ is $1/2^n$, and the expected number of steps is 2^n .
11. a. We assume that it is possible to compute discrete logarithms in H and $y^t \in H$.
 b. The verification condition in ElGamal's signature scheme is $g^m = y^r r^s$.

$$\begin{aligned}
y^r r^s &= y^t r^s = g^{tz} r^s \\
&= g^{tz} t^{(p-3)(m-tz)/2} = g^{tz} \left(t^{(p-1)/2} t^{-1} \right)^{m-tz} \\
&= g^{tz} (-t^{-1})^{m-tz} = g^{tz} g^{m-tz} = g^m.
\end{aligned}$$

Note that $gt = p - 1 = -1 \pmod p$ implies $t = -g^{-1}$ and $g = -t^{-1}$ and that

$$t^{(p-1)/2} = (-g)^{-(p-1)/2} = (-1)^{-(p-1)/2} g^{-(p-1)/2} = 1 \cdot (-1) = -1$$

($g^{-(p-1)/2} = -1$, since g is a primitive root).

c. The above attack does not work in the DSA signature scheme.

4. Cryptographic Protocols

1. With this protocol the simple man-in-the-middle attack does not work. A more sophisticated attack is necessary. If adversary Eve selects e and declares y_A^e as her public key, a man-in-the-middle attack works:
 - a. Eve intercepts c and forwards it unchanged to Bob.
 - b. Eve intercepts d and forwards d^e to Alice.
 Then Alice computes $k = d^{e x_A} y_B^a = g^{b e x_A} g^{a x_B}$. She believes that she shares k with Bob. Whereas Bob believes that he shares $k = c^{x_B} y_E^b = g^{a x_B} g^{b e x_A}$ with Eve. Eve cannot compute the session key k . However, she can masquerade as Alice.

2. Protocol 4.1.

OneOfTwoSquareRoots(x_1, x_2)

Case: Peggy knows a square root y_1 of x_1 (the other case follows analogously):

1. Peggy chooses at random $r_1, r_2 \in \mathbb{Z}_n^*$ and $e_2 \in \{0, 1\}$ and sets $a = (a_1, a_2) = (r_1^2, r_2^2 x_2^{e_2})$. Peggy sends a to Vic.
2. Vic chooses at random $e \in \{0, 1\}$. Vic sends e to Peggy.
3. Peggy computes

$$e_1 = e \oplus e_2,$$

$$b = (b_1, b_2) = (r_1 y_1^{e_1}, r_2)$$

and sends b, e_1, e_2 to Vic.

4. Vic accepts, if and only if

$$e = e_1 \oplus e_2,$$

$$b_1^2 = a_1 x_1^{e_1}, b_2^2 = a_2 x_2^{e_2}.$$

The completeness, soundness and zero-knowledge properties are analogously proven as in Protocol 4.5.

3. a. Let $x \in \text{QNR}_n^{+1}$. $a = r^2 x^\sigma \in \text{QR}_n \iff \sigma = 0$. Thus $\sigma = \tau$ and Vic will accept.
- b. Let $x \in \text{QR}_n$. Then $a = r^2 x^\sigma \in \text{QR}_n$, for $\sigma \in \{0, 1\}, r \in \mathbb{Z}_n^*$. Thus τ is always 1 and $\text{prob}(\sigma = \tau) = 1/2$. Thus a dishonest Peggy can convince Vic with probability $1/2$ if $x \in \text{QR}_n$.
- c. Let V^* be a dishonest verifier defined by the following

Protocol 4.2.

PQR $_n$

1. V^* chooses at random $r \in \mathbb{Z}_n^*$ with $\left(\frac{x}{n}\right) = 1$ and sends $a = r$ to Peggy.
2. Peggy computes $\tau := \begin{cases} 0 & \text{if } a \in \text{QR}_n \\ 1 & \text{if } a \notin \text{QR}_n \end{cases}$ and sends τ to Vic.

3. V^* outputs τ .

Note $\tau = 0$ if $r \in \text{QR}_n$ and $\tau = 1$ if $r \notin \text{QR}_n$. Thus V^* can decide after interaction with Peggy, whether a randomly chosen r is a quadratic residue. Without Peggy's help he cannot do this according to the quadratic residuosity assumption (see Section 4.3.1 and Definition 6.11).

d. **Algorithm 4.3.**

int $S(\text{int } x)$

- 1 select $r \in \mathbb{Z}_n^*$ and $\sigma \in \{0, 1\}$ uniformly at random
- 2 return $(\tilde{a}, \tilde{\tau}) \leftarrow (r^2 x^\sigma, \sigma)$

By construction, the random variables $S(x)$ and $(P, V)(x)$ are identically distributed for $x \in \text{QNR}_n$.

- e. Vic proofs to Peggy after step 1 that he knows a square root of a or of a/x by using the protocol of Exercise 2. He can only succeed, if he followed the protocol in step 1. Thus he is a honest verifier and d) applies.
4. The idea is as in Exercise 3e). The verifier proves that he follows the protocol in step 1, i.e., that he sends a message which he encrypted with the public key. For this purpose, he shows that he knows the e -th root of the message he transmitted.

To show that a prover Peggy knows the e -th root x of y , the following protocol may be used.

Protocol 4.4.

e -th root(y)

1. Peggy chooses at random $r \in \mathbb{Z}_n^*$ and sets $a = r^e$. Peggy sends a to Vic.
2. Vic chooses at random $\sigma \in \{0, 1\}$. Vic sends σ to Peggy.
3. Peggy computes $b = rx^\sigma$ and sends b to Vic, i.e., Peggy sends r , if $e = 0$, and rx , if $\sigma = 1$.
4. Vic accepts, if and only if $b^e = ay^\sigma$.

The completeness, soundness and zero-knowledge properties are analogously proven as in Protocol 4.5.

5. a. Alice commits to 0, if $c \in \text{QR}_n$ and to 1, if $c \notin \text{QR}_n$.
Note: $c \in \text{QR}_n \iff -c \notin \text{QR}_n$.
- b. $c_1 c_2 = r_1^2 r_2^2 (-1)^{b_1 + b_2 \bmod 2} = (r_1 r_2)^2 (-1)^{b_1 \oplus b_2}$.
- c. c_1 and c_2 commit to the same value, if $c_1 c_2 \in \text{QR}_n$. They commit to different values, if $c_1 c_2 \notin \text{QR}_n$. Both cases can be proven by zero-knowledge proofs (see Section 4.2.4 and Exercise 3).
6. The access structure can be realized, if P_1 gets three shares, P_2 two shares and P_3, P_4, P_5 and P_6 each get one share in a $(5, n)$ -Shamir threshold scheme.

7. Assume P_i has p_i shares of a (t, n) -Shamir threshold scheme. Then $p_1 + p_2 \geq t$ and $p_3 + p_4 \geq t$. Thus $p_1 + p_2 + p_3 + p_4 \geq 2t$. $p_1 + p_3 < t$ implies $p_2 + p_4 \geq t$. Thus $\{P_1, P_3\}$ or $\{P_2, P_4\}$ are also able to reconstruct the secret.
8. We use the notations of Section 4.4. The encryption scheme allows to encrypt every message $m = g^v, 0 \leq v \leq q - 1$. Thus, a voter could encrypt up to $(q - 1)/2$ "yes-" or "no-votes". If an authority posts $w_j g$ or $w_j g^{-1}$, the tally is decreased or increased by $\lambda_{i,J}$.
9. We write g_1, g_2 instead of g, h (below, we denote by h a hash function).

Protocol 4.5.

OneOfTwoPairs($g_1, g_2, (y_1, z_1), (y_2, z_2)$)

Case: Peggy knows $\log_{g_1} y_1 = \log_{g_2} z_1 = x$ (the other case follows analogously):

1. Peggy chooses at random r_1, r_2 and $d_2 \in \{0, \dots, q - 1\}$ and sets $a = (a_1, a_2, a_3, a_4) = (g_1^{r_1}, g_2^{r_1}, g_1^{r_2} y_2^{d_2}, g_2^{r_2} z_2^{d_2})$. Peggy sends a to Vic.
2. Vic chooses $c \in \{0, \dots, q - 1\}$ uniformly at random. Vic sends c to Peggy.
3. Peggy computes

$$d_1 = c - d_2 \text{ mod } q,$$

$$b = (b_1, b_2) = (r_1 - d_1 x, r_2)$$

and sends (b_1, b_2, d_1, d_2) to Vic.

4. Vic accepts, if and only if

$$c = d_1 + d_2 \text{ mod } q,$$

$$a_1 = g_1^{b_1} y_1^{d_1},$$

$$a_2 = g_2^{b_1} z_1^{d_1},$$

$$a_3 = g_1^{b_2} y_2^{d_2},$$

$$a_4 = g_2^{b_2} z_2^{d_2}.$$

The prover Peggy can convert this interactive proof into a non-interactive proof.

$$(d_1, d_2, b_1, b_2) = \text{OneOfTwoPairs}_h(g_1, g_2, (y_1, z_1), (y_2, z_2))$$

She proceeds in step 1 as before. Then, she computes the challenge $c = h(g_1 \| g_2 \| y_1 \| z_1 \| y_2 \| z_2 \| a_1 \| a_2 \| a_3 \| a_4)$, by using a collision-resistant hash function h .

The verification condition is

$$d_1 + d_2 = h(g_1 \| g_2 \| y_1 \| z_1 \| y_2 \| z_2 \| g_1^{b_1} y_1^{d_1} \| g_2^{b_1} z_1^{d_1} \| g_1^{b_2} y_2^{d_2} \| g_2^{b_2} z_2^{d_2}).$$

10. Voter V_j can duplicate the vote $c_i = (c_{i,1}, c_{i,2})$ of voter V_i . For this purpose, he selects α and sets $c_j = (c_{i,1}g^\alpha, c_{i,2}h^\alpha)$. He has to prove that his vote is a correctly formed one, by the protocol *OneOfTwoPairs* from Exercise 9. We first discuss the case, where the interactive version of the proof is applied.

- a. Voter V_j can derive from voter V_i 's proof

$$(a, d, b) = \text{OneOfTwoPairs}(g, h, (y_1, z_1), (y_2, z_2)),$$

where

$$\begin{aligned} y_1 &= c_{i,1}, \quad z_1 = c_{i,2}g, \quad y_2 = c_{i,1}, \quad z_2 = c_{i,2}g^{-1}, \\ a &= (a_1, a_2, a_3, a_4), \\ d &= (d_1, d_2), \quad b = (b_1, b_2), \end{aligned}$$

the proof

$$(\tilde{a}, \tilde{d}, \tilde{b}) = \text{OneOfTwoPairs}(g, h, (\tilde{y}_1, \tilde{z}_1), (\tilde{y}_2, \tilde{z}_2)),$$

where

$$\begin{aligned} \tilde{y}_1 &= y_1g^\alpha, \quad \tilde{z}_1 = z_1h^\alpha \\ \tilde{y}_2 &= y_2g^\alpha, \quad \tilde{z}_2 = z_2h^\alpha \\ \tilde{a} &= a, \quad \tilde{d} = d, \\ \tilde{b} &= (b_1 - d_1\alpha, b_2 - d_2\alpha). \end{aligned}$$

- b. With the non-interactive proof, the attack does not work. Replacing the argument $(y_i, z_i, i = 1, 2)$ of the hash function will cause a different output. Note, the hash function is assumed to be collision resistant. To duplicate a vote, an identical copy of the ballot must be used. However, it will be detected, if a ballot is posted twice.

11. Protocol 4.6.

BlindRSASig(m)

1. Vic randomly chooses $r \in \mathbb{Z}_n^*$, computes $\bar{m} = r^\epsilon m$ and sends it to Peggy.
 2. Peggy computes $\sigma = \bar{m}^d$ and sends it to Vic.
 3. Vic computes σr^{-1} and gets the signature of m .
12. a. $ry^r g^{-s} = mg^k g^{xr} g^{-(xr+k)} = m$.
- b. Choose any r, s with $1 \leq r \leq p-1$ and $1 \leq s < q-1$ and let $m := ry^r g^{-s}$. Then (m, r, s) is a signed message. This kind of attack is always possible, if the message can be recovered from the signature, as in the basic Nyberg-Rueppel scheme.
- c. Use a collision-resistant hash function h and hash before encrypting, or, if you want to preserve the message recovery property, apply a suitable bijective redundancy function R to the message to be signed (see [MenOorVan96]).

- d. Let (m, r, s) be a valid signature. Without the first check, an attacker may sign messages \tilde{m} of his choice. He computes $g^k = rm^{-1}$ by the extended Euclidean algorithm. Then, he uses the Chinese remainder theorem to determine a $\tilde{r} \in \mathbb{Z}$ with $\tilde{r} \equiv \tilde{m}g^k \pmod{p}$ and $\tilde{r} \equiv r \pmod{q}$. Then $(\tilde{m}, \tilde{r}, s)$ passes the verification, if $1 \leq \tilde{r} \leq p-1$ is not checked.

13. a. (m, r, s) is a signed message:

$$\begin{aligned} ry^r g^{-s} &= \tilde{r}\alpha g^{rx} g^{-(\tilde{s}\alpha+\beta)} \\ &= mg^{\tilde{k}(\alpha-1)} g^{\tilde{k}} g^\beta g^{rx} g^{-(\tilde{s}\alpha+\beta)} \\ &= mg^{\alpha\tilde{k}} g^\beta g^{rx} g^{-(\tilde{s}\alpha+\beta)} \\ &= mg^{\alpha(\tilde{k}-\tilde{s})} g^{rx} \\ &= mg^{-\alpha\tilde{r}x} g^{rx} \\ &= m. \end{aligned}$$

- b. The protocol is blind: The transcript $(\tilde{a}, \tilde{m}, \tilde{r}, \tilde{s})$ is transformed into the signed message (m, r, s) by

$$\begin{aligned} m &= \tilde{m}\tilde{a}^{-(\alpha-1)} g^{-\beta} \alpha, \\ r &= \tilde{r}\alpha, \\ s &= \tilde{s}\alpha + \beta. \end{aligned}$$

The message m is uniquely determined by r and s ($m = ry^r g^{-s}$). On the other hand, α and β are uniquely determined by r, s and \tilde{r}, \tilde{s} . The transcript $(\tilde{a}, \tilde{m}, \tilde{r}, \tilde{s})$ can be transformed to any (r, s) and hence, to any signed message (m, r, s) . Every signed message (m, r, s) is equally likely to be the transformation of the transcript $(\tilde{a}, \tilde{m}, \tilde{r}, \tilde{s})$, if α and β are chosen at random. Thus the signature is really blind.

14. **Protocol 4.7.**

ProofRep(g_1, g_2, y)

1. Peggy randomly chooses $r_1, r_2 \in \mathbb{Z}_q$, computes $a = g^{r_1} g^{r_2}$ and sends it to Vic.
2. Vic chooses at random $c \in \mathbb{Z}_q$ and sends it to Peggy.
3. Peggy computes $b_i = r_i - cx_i, i = 1, 2$, and sends (b_1, b_2) to Vic.
4. Vic accepts the proof, if

$$a = g_1^{b_1} g_2^{b_2} y^c,$$

otherwise, he rejects it.

15. **Protocol 4.8.**

BlindRepSig_h(m)

1. Peggy randomly chooses $\bar{r}_1, \bar{r}_2 \in \mathbb{Z}_q$, computes $\bar{a} = g_1^{\bar{r}_1} g_2^{\bar{r}_2}$ and sends it to Vic.
2. Vic chooses at random $u \in \mathbb{Z}_q^*$, $v_1, v_2, w \in \mathbb{Z}_q$ and computes

$$\begin{aligned} a &= \bar{a}^u g_1^{v_1} g_2^{v_2} y^w, \\ c &= h(m \| a), \bar{c} = (c - w)u^{-1}. \end{aligned}$$

Vic sends \bar{c} to Peggy.

3. Peggy computes $\bar{b} = (\bar{b}_1, \bar{b}_2) = (\bar{r}_1 - \bar{c}x_1, \bar{r}_2 - \bar{c}x_2)$ and sends it to Vic.
4. Vic verifies whether

$$\bar{a} = g_1^{\bar{b}_1} g_2^{\bar{b}_2} y^{\bar{c}},$$

computes $b = (b_1, b_2) = (u\bar{b}_1 + v_1, u\bar{b}_2 + v_2)$ and gets the signature $\sigma(m) = (c, b)$ of m .

The verification condition for a signature (c, b) is $c = h(m \| g_1^{b_1} g_2^{b_2} y^c)$.

5. Probabilistic Algorithms

1. The desired Las Vegas algorithm works as follows:

Repeat

1. Compute $y = A(x)$.
2. Check by $D(x, y)$, whether y is a correct solution for input x .
3. If the check yields 'yes', then return y and stop. Otherwise, go back to 1.

The expected number of iterations is $1/\text{prob}(A(x) \text{ correct})$ (by Lemma B.12) and hence $\leq P(|x|)$. The binary length of an output y is bounded by $R(|x|)$. Thus, the running time of $D(x, y)$ is bounded by $S(|x| + R(|x|))$.

2. We define the algorithm \tilde{A} on input x as follows:

- a. Let $t(x) := P(|x|)^2 Q(|x|)$.
- b. Compute $A(x)$ $t(x)$ -times, and obtain the results $b_1, \dots, b_{t(x)} \in \{0, 1\}$.
- c. Let

$$\tilde{A}(x) := \begin{cases} +1 & \text{if } \frac{1}{t(x)} \sum_{i=1}^{t(x)} b_i \geq a \\ 0 & \text{if } \frac{1}{t(x)} \sum_{i=1}^{t(x)} b_i < a. \end{cases}$$

From Corollary B.17 applied to the $t(x)$ independent computations of $A(x)$, we get for $x \in \mathcal{L}$

$$\text{prob} \left(\frac{1}{t(x)} \sum_{i=1}^{t(x)} b_i < a \right) < \frac{P(|x|)^2}{4t(x)} < \frac{1}{Q(|x|)},$$

and for $x \notin \mathcal{L}$

$$\text{prob} \left(\frac{1}{t(x)} \sum_{i=1}^{t(x)} b_i \geq a \right) < \frac{P(|x|)^2}{4t(x)} < \frac{1}{Q(|x|)}.$$

3.
 - a. The probability that $A(x)$ returns at least one 1 during t executions of $A(x)$ is 0 if $x \notin \mathcal{L}$, and $> 1 - (1 - 1/Q(|x|))^t$ if $x \in \mathcal{L}$. For $t \geq \ln(2)Q(|x|)$, we have $(1 - 1/Q(|x|))^t \leq 1/2$ (see proof of Proposition 5.7).
 - b. Consider an \mathcal{NP} -problem \mathcal{L} and a deterministic polynomial algorithm $M(x, y)$ that answers the membership problem for \mathcal{L} with certificates of length $\leq L(|x|)$. Selecting $y \in \{0, 1\}^{L(|x|)}$ by coin tosses and calling $M(x, y)$, we get a probabilistic polynomial algorithm $A(x)$ with deterministic extension $M(x, y)$.
Conversely, a probabilistic polynomial algorithm A that decides the membership in \mathcal{L} yields a deterministic M .
These considerations show that a problem \mathcal{L} is in \mathcal{NP} if and only if there is a probabilistic polynomial algorithm A with values in $\{0, 1\}$,

such that $\text{prob}(A(x) = 1) > 0$ if $x \in \mathcal{L}$, and $\text{prob}(A(x) = 1) = 0$ if $x \notin \mathcal{L}$.

Now, the inclusion $\mathcal{RP} \subseteq \mathcal{NP}$ is obvious (to obtain this inclusion, only the 'conversely'- direction of our considerations is necessary).

4. Let $A(x)$ be a Las Vegas algorithm for the membership in a \mathcal{ZPP} -problem \mathcal{L} , and let $P(|x|)$ be a polynomial bound for the expected running time of A . We define a Monte Carlo algorithm $\tilde{A}(x)$ as follows. We call $A(x)$. If $A(x)$ returns after less than $P(|x|)$ steps, we set $\tilde{A}(x) = A(x)$. Otherwise, let $\tilde{A}(x) = 0$. Then \tilde{A} is an algorithm for the membership in \mathcal{L} , as it is required for \mathcal{RP} -problems.
5. Proposition 5.6 can be improved, such that the probability of success of the algorithm \tilde{A} is exponentially close to 1. More precisely:
By repeating the computation $A(x)$ and by returning the most frequent result, we get a probabilistic polynomial algorithm \tilde{A} , such that

$$\text{prob}(\tilde{A}(x) = f(x)) > 1 - 2^{-Q(|x|)} \text{ for all } x \in X.$$

The proof is completely analogous to the proof of Proposition 5.6. The Chernoff bound is used instead of Proposition 5.6 (which is a consequence of the weak law of large numbers B.16). The Chernoff bound implies that

$$\text{prob} \left(\sum_{j=1}^t S_j > \frac{t}{2} \right) \geq 1 - 2e^{-\frac{t}{P(|x|)^2}}.$$

For $t > \ln(2)P(|x|)^2(Q(|x|) + 1)$, we get the desired result.

6. One-Way Functions and the Basic Assumptions

1. a. Let $\tilde{I}_k := \{n \in \mathbb{N} \mid n = pq, p, q \text{ distinct primes}, |p| = |q| = k\}$. The set of keys of security parameter k is $I_k = \{(n, e) \mid n \in \tilde{I}_k, e \in \mathbb{Z}_{\varphi(n)}^*\}$. Let p_k be the uniform distribution on I_k and let q_k be the distribution $i \leftarrow S(1^k)$, given by S . Then

$$p_k(n, e) = \frac{1}{|\tilde{I}_k|} \cdot \frac{1}{\text{aver}(|\mathbb{Z}_{\varphi(n)}^*|)} \quad \text{and} \quad q_k(n, e) = \frac{1}{|\tilde{I}_k|} \cdot \frac{1}{|\mathbb{Z}_{\varphi(n)}^*|},$$

where $\text{aver}(|\mathbb{Z}_{\varphi(n)}^*|)$ is the average value taken over $n \in \tilde{I}_k$. As we observed in the proof of Proposition 6.6 (referring to Appendix A.2), $\varphi(x) > \frac{x}{6 \log(|x|)}$. Hence,

$$\varphi(n) > |\mathbb{Z}_{\varphi(n)}^*| = \varphi(\varphi(n)) > \frac{\varphi(n)}{c \log(k)}$$

(c a constant). This implies $q_k(n, e) \leq c \cdot \log(k) \cdot p_k(n, e)$. In particular, q_k is polynomially bounded by p_k .

b. Analogous to a).

2. The number of primes of length k is of order $2^k/k$ (by the Prime Number Theorem A.68). Thus, we expect to get a prime after $O(k)$ iterations if we randomly choose k -bit strings and apply a probabilistic primality test (see Lemma B.12). A probabilistic primality test takes $O(k^3)$ steps (step = binary operation) and therefore, the expected running time to generate a random prime of length k is $O(k^4)$. To choose a random $e \in \mathbb{Z}_{\varphi(n)}$ and to check, whether it is a unit (by Euclid's algorithm A.4), takes $O(k^3)$ steps. The probability of getting a unit is $\varphi(\varphi(n))/\varphi(n)$, with $\varphi(n) = (p-1)(q-1)$. Let $d := \lceil \text{average}_{n \in I_k} \varphi(n)/\varphi(\varphi(n)) \rceil$. In the uniform sampling algorithm of Proposition 6.8, we expect to get a key after generating d moduli $n = pq$ and d exponents. Applying the admissible key generator from Exercise 1, we expect to get a key after generating one modulus and d exponents. Thus, the expected running time of the uniform key generator of Proposition 6.8 is about d -times the expected running time of the admissible key generator from Exercise 1. We have $d \leq 6 \log(2k)$ (Appendix A.2).
3. f is certainly not a strong one-way function: Half of the elements of X_j are even. For every $(x, y) \in D_n$, with x or y is even and $xy < 2^{n-1}$, a pre-image $(2, xy/2)$ of $f_n(x, y)$ is immediately computed. Let $\tilde{D}_n := \{(x, y) \in D_n \mid x, y \text{ are primes with } |x| = |y| = \lfloor n/2 \rfloor\}$. We have (by the Prime Number Theorem A.68)

$$|\tilde{D}_n| \approx \left(\frac{2^{\lfloor n/2 \rfloor - 1}}{\lfloor n/2 \rfloor - 1} \right)^2 \geq \frac{2^{n-3}}{((n-1)/2)^2} \geq \frac{2^{n-1}}{n^2} = \frac{2^n}{2n^2}.$$

On the other hand, $|D_n| = \sum_{j=2}^{n-2} 2^j 2^{n-j} < n2^n$ and hence

$$\frac{|\tilde{D}_n|}{|D_n|} \geq \frac{1}{2n^3}.$$

By the factoring assumption, the pre-image of xy cannot be efficiently computed with a non-negligible probability for $(x, y) \in \tilde{D}_n$. Thus, the probability of success of an adversary algorithm is $\leq 1 - 1/2n^3$.

4. Let A_1 be the algorithm that calls A and then returns the difference $(a_1 - a'_1, \dots, a_r - a'_r)$ of A 's outputs. As we already observed in the proof of Proposition 4.21, A_1 computes a non-trivial representation $1 = \prod_{j=1}^r g_j^{e_j}$ of 1 if and only if A computes two distinct representations $\prod_{j=1}^r g_j^{a_j} = \prod_{j=1}^r g_j^{a'_j}$ of the same element in G_q .

To compute the discrete logarithm of an element $y \in G_q$ with respect to g , we use the algorithm B (see the algorithm given in the proof of Proposition 4.21):

Algorithm 6.1.

```

int B(int p, q, g, y)
1  if y = 1
2    then return 0
3  else select i ∈ {1, ..., r} and
4         u_j ∈ {1, ..., q - 1}, 1 ≤ j ≤ r, uniformly at random
5         g_i ← y^{u_i}
6         g_j ← g^{u_j}, 1 ≤ j ≠ i ≤ r, is chosen at random
7         (a_1, ..., a_r) ← A(g_1, ..., g_r)
8         if a_i ≠ 0 mod q
9           then return x ← -(a_i u_i)^{-1} (∑_{j≠i} a_j u_j) mod q
10        else return 0
    
```

If A_1 returns a non-trivial representation and if $a_i \neq 0$ (modulo q), then

$$y^{-u_i a_i} = \prod_{j \neq i} g^{a_j u_j},$$

and B correctly returns $\log_g(y)$ of y with respect to the base g .

If $y \neq 1$, then y is a generator of G_q and y^{u_i} is an element which is randomly and uniformly chosen from $G_q \setminus \{1\}$, and this random choice is independent of the choice of i . If A_1 returns a non-trivial representation of 1, then at least one $a_j \neq 0 \pmod q$ and therefore, the probability that we get a position i with $a_i \neq 0 \pmod q$ by the random choice of i , is $\geq 1/r \geq 1/T(|p|)$. Thus,

$$\begin{aligned} \text{prob}(B(p, q, g, y) = \log_g(y)) \\ \geq \text{prob}(A(p, q, g_1, \dots, g_r) = (a_1, \dots, a_r) \neq 0, \quad 1 = \prod_{j=1}^r g_j^{a_j} : \end{aligned}$$

$$g_j \stackrel{u}{\leftarrow} G_q \setminus \{1\}, \quad 1 \leq j \leq r$$

$$\begin{aligned} & \cdot \frac{1}{T(|p|)} \\ & \geq \frac{1}{P(|p|)} \cdot \frac{1}{T(|p|)}, \end{aligned}$$

for every $g \in G_q \setminus \{1\}$, $y \in G_q$ and $(p, q) \in \mathcal{K}$. By repeating the computation $B(p, q, g, y)$ for a sufficiently large (but polynomial in $|p|$) number of times and each time checking whether the output is the desired logarithm, we get a probabilistic polynomial algorithm $\tilde{A}(p, q, g, y)$ with $\text{prob}(\tilde{A}(p, q, g, y) = \log_g(y)) \geq 1 - 2^{-Q(|p|)}$ (Proposition 5.7).

5. Let $I_k := \{(n, e) \mid n = pq, p, q \text{ distinct primes}, |p| = |q| = k, e \in \mathbb{Z}_{\varphi(n)}^*\}$ be the set of public RSA keys with security parameter k . By Exercise 1, the RSA assumption remains valid if we replace $(n, e) \stackrel{u}{\leftarrow} I_k$ by $n \stackrel{u}{\leftarrow} J_k, e \stackrel{u}{\leftarrow} \mathbb{Z}_{\varphi(n)}^*$. In the following sequence of distributions, each distribution polynomially bounds its successor.

- a. $n \stackrel{u}{\leftarrow} J_k, e \stackrel{u}{\leftarrow} \mathbb{Z}_{\varphi(n)}^*$
- b. $n \stackrel{u}{\leftarrow} J_k, e \stackrel{u}{\leftarrow} \{f < 2^{2k} \mid f \text{ prime to } \varphi(n)\}$
- c. $n \stackrel{u}{\leftarrow} J_k, \tilde{p} \stackrel{u}{\leftarrow} \{f \in \text{Primes}_{\leq 2k} \mid f \text{ does not divide } \varphi(n)\}$

The example after Definition B.25, shows that a) bounds b) (consider the map $x \mapsto x \bmod \varphi(n)$). b) bounds c), since the number of primes of binary length $\leq 2k$ is about $\frac{2^{2k}}{k^2}$ (Theorem A.68). By Proposition B.26, we conclude that the RSA assumption remains valid if we replace $(n \stackrel{u}{\leftarrow} J_k, e \stackrel{u}{\leftarrow} \mathbb{Z}_{\varphi(n)}^*)$ by $(n \stackrel{u}{\leftarrow} J_k, \tilde{p} \stackrel{u}{\leftarrow} \{f \in \text{Primes}_{\leq 2k} \mid f \text{ does not divide } \varphi(n)\})$. By Lemma B.24, this distribution - we call it q - can be replaced by $(n \stackrel{u}{\leftarrow} J_k, \tilde{p} \stackrel{u}{\leftarrow} \text{Primes}_{\leq 2k})$, since both distributions are polynomially close. Namely, we have for large k (up to some constant)

$$|\{f \in \text{Primes}_{\leq 2k} \mid f \text{ does not divide } \varphi(n)\}| \geq |\text{Primes}_{\leq 2k}| - \log_2(2k),$$

hence by Theorem A.68

$$\begin{aligned} & \left| q(\tilde{p}, n) - \frac{1}{|J_k|} \cdot \frac{1}{|\text{Primes}_{\leq 2k}|} \right| \\ & \leq \frac{1}{|J_k|} \cdot \frac{1}{|\text{Primes}_{\leq 2k}|} \cdot \left(\frac{|\text{Primes}_{\leq 2k}|}{|\text{Primes}_{\leq 2k}| - \log_2(2k)} - 1 \right) \\ & \approx \frac{1}{|J_k|} \cdot \frac{1}{|\text{Primes}_{\leq 2k}|} \cdot \left(\frac{\frac{2^{2k}}{2k}}{\frac{2^{2k} - 2k \log_2(2k)}{2k}} - 1 \right) \\ & \approx \frac{1}{|J_k|} \cdot \frac{1}{|\text{Primes}_{\leq 2k}|} \cdot \frac{2k \log_2(2k)}{2^{2k}} \approx \frac{k}{2^k} \cdot \frac{k}{2^k} \cdot \frac{2k}{2^{2k}} \cdot \frac{2k \log_2(2k)}{2^{2k}} \leq \frac{k^5}{2^{6k}}. \end{aligned}$$

Since the number of tuples (\tilde{p}, n) is of order $O(\frac{2^{4k}}{k^4})$, the polynomial closeness follows.

Finally, by Theorem A.70, we have for a prime \tilde{p} (up to some constant)

$$|\{f \in \text{Primes}_k \mid \tilde{p} \text{ divides } f - 1\}| \approx \frac{1}{\tilde{p} - 1} \frac{2^k}{k} \leq \frac{2^k}{2k},$$

hence

$$|J_{k,\tilde{p}}| \geq \frac{2^{2k}}{4k^2} \text{ and then } 4 \cdot |J_{k,\tilde{p}}| \geq |J_k| \approx \frac{2^{2k}}{k^2}.$$

We see that $(n \stackrel{u}{\leftarrow} J_k, \tilde{p} \stackrel{u}{\leftarrow} \text{Primes}_{\leq 2k})$ polynomially bounds $(\tilde{p} \stackrel{u}{\leftarrow} \text{Primes}_{\leq 2k}, n \stackrel{u}{\leftarrow} J_{k,\tilde{p}})$. This finishes the proof.

6. Let $b \in \{0, 1\}$. Assume that there is a positive polynomial P , such that

$$\text{prob}(B_i(x) = b : i \leftarrow K(1^k), x \stackrel{u}{\leftarrow} D_i) - \frac{1}{2} > \frac{1}{P(k)},$$

for infinitely many k . Then the constant algorithm $A(i, y)$ that always returns b successfully computes the hard-core bit

$$\begin{aligned} \text{prob}(A(i, f_i(x)) = B_i(x) : i \leftarrow K(1^k), x \stackrel{u}{\leftarrow} D_i) \\ = \text{prob}(B_i(x) = b : i \leftarrow K(1^k), x \stackrel{u}{\leftarrow} D_i) \geq \frac{1}{2} + \frac{1}{P(k)}, \end{aligned}$$

a contradiction.

7. Assume there is an algorithm A with

$$\begin{aligned} \text{prob}(A(i, f_i(x), B_i(x)) = 1 : i \leftarrow K(1^k), x \stackrel{u}{\leftarrow} D_i) \\ - \text{prob}(A(i, f_i(x), z) = 1 : i \leftarrow K(1^k), x \stackrel{u}{\leftarrow} D_i, z \stackrel{u}{\leftarrow} \{0, 1\}) > \frac{1}{P(k)} \end{aligned}$$

for some positive polynomial P and for k in an infinite subset \mathcal{K} of \mathbb{N} (Replacing A by $1 - A$, if necessary, we may omit the absolute value). Let \tilde{A} be the following algorithm with inputs $i \in I, y \in R_i$:

- a. Randomly choose a bit $b \stackrel{u}{\leftarrow} \{0, 1\}$.
- b. If $A(i, y, b) = 1$, then return b , else return $1 - b$.

Applying Lemma B.13 we get

$$\begin{aligned} \text{prob}(\tilde{A}(i, f_i(x)) = B_i(x) : i \leftarrow K(1^k), x \stackrel{u}{\leftarrow} D_i) \\ = \frac{1}{2} + \text{prob}(\tilde{A}(i, f_i(x)) = b : i \leftarrow K(1^k), x \stackrel{u}{\leftarrow} D_i \mid B_i(x) = b) \\ \quad - \text{prob}(\tilde{A}(i, f_i(x)) = b : i \leftarrow K(1^k), x \stackrel{u}{\leftarrow} D_i) \\ = \frac{1}{2} + \text{prob}(A(i, f_i(x), B_i(x)) = 1 : i \leftarrow K(1^k), x \stackrel{u}{\leftarrow} D_i) \\ \quad - \text{prob}(A(i, f_i(x), b) = 1 : i \leftarrow K(1^k), x \stackrel{u}{\leftarrow} D_i, b \stackrel{u}{\leftarrow} \{0, 1\}) \\ > \frac{1}{2} + \frac{1}{P(k)}. \end{aligned}$$

for the infinitely many $k \in \mathcal{K}$. Hence, B is not a hard-core predicate. Conversely, if $\tilde{A}(i, y)$ is a probabilistic polynomial algorithm with

$$\text{prob}(\tilde{A}(i, f_i(x)) = B_i(x) : i \leftarrow K(1^k), x \xleftarrow{u} D_i) > \frac{1}{2} + \frac{1}{P(k)}$$

for infinitely many k , then the algorithm A with

$$A(i, y, z) := \begin{cases} 1 & \text{if } z = \tilde{A}(i, y), \\ 0 & \text{else.} \end{cases}$$

successfully distinguishes between the distributions.

8. The analogous proposition is:

The following statements are equivalent.

- a. For every probabilistic polynomial algorithm A with inputs $i \in I$, $x \in X_i$ and output in $\{0, 1\}$ and every positive polynomial P , there is a $k_0 \in \mathbb{N}$, such that for all $k \geq k_0$

$$\left| \text{prob}(A(i, x) = 1 : i \leftarrow I_k, x \xleftarrow{P_i} X_i) - \text{prob}(A(i, x) = 1 : i \leftarrow I_k, x \xleftarrow{Q_i} X_i) \right| \leq \frac{1}{P(k)}.$$

- b. For every probabilistic polynomial algorithm A with inputs $i \in I$, $x \in X_i$ and output in $\{0, 1\}$ and all positive polynomials Q, R there is a $k_0 \in \mathbb{N}$, such that for all $k \geq k_0$

$$\begin{aligned} & \text{prob}(\{i \in I_k \mid |\text{prob}(A(i, x) = 1 : x \xleftarrow{P_i} X_i) \\ & \quad - \text{prob}(A(i, x) = 1 : x \xleftarrow{Q_i} X_i)| > \frac{1}{Q(k)}\}) \\ & \leq \frac{1}{R(k)}. \end{aligned}$$

The proof now runs in the same way as the proof of Proposition 6.17. The main difference is that we need an algorithm $\text{Sign}(i)$ which computes the sign of

$$\text{prob}(A(i, x) = 1 : x \xleftarrow{P_i} X_i) - \text{prob}(A(i, x) = 1 : x \xleftarrow{Q_i} X_i)$$

with high probability if the absolute value of this difference is $\geq 1/\tilde{T}(k)$ (with \tilde{T} a polynomial). This algorithm is constructed analogously. We use the fact that the probabilities can be approximately computed with high probability by a probabilistic polynomial algorithm (Proposition 6.18).

9. see [GolMic84].

7. Bit Security of One-Way Functions

1.

$$\begin{aligned}
 17 &\in \text{QR}_p, \\
 \text{PSqrt}(17) &= 13 \notin \text{QR}_p, 13 \cdot 2^{-1} = 16 \\
 \text{PSqrt}(16) &= 4 \in \text{QR}_p \\
 \text{PSqrt}(4) &= 2 \notin \text{QR}_p, 2 \cdot 2^{-1} = 1 \\
 \text{PSqrt}(1) &= 1 \in \text{QR}_p
 \end{aligned}$$

Thus we have $\text{Log}_{p,g}(17) = 01010$ (in binary encoding).

2. **Algorithm 7.1.**

```

int BinSearchLog(int p, g, y)
1  int : l, r, i
2  l ← 0; r ← p - 1
3  while l ≤ r do
4      i ← div(l + r)
5      if A1(p, g, y) = 1
6          then l ← i
7      else r ← i + 1
8      y ← y2
9  return l

```

3. a. We compute the t least-significant bits as in the proof of Proposition 7.5. Let $k = |p|$, $y = g^{x_{k-1}\dots x_0}$, $x_i \in \{0, 1\}$, $i = 0, \dots, k-1$. The bit x_0 is 0, if and only if $y \in \text{QR}_p$ (Proposition A.49). This condition can be tested with the criterion of Euler for quadratic residuosity (Proposition A.52).

We replace y by yg^{-1} , if $x_0 = 1$. Thus, we can assume $x_0 = 0$. We get the square roots $y_1 = g^{x_{k-1}\dots x_1}$ and $y_2 = g^{x_{k-1}\dots x_1 + (p-1)/2}$ of y . Since $p-1 = 2^t q$, q odd, the t least-significant bits of $p-1$ are 0. $\log y_1$ and $\log y_2$ coincide in the $t-1$ least-significant bits ($t \geq 1$). If $t \geq 2$, we can continue with both square roots.

Algorithm 7.2.

```

int A(int p, g, x)
1  d ← ε
2  for c ← 0 to t - 1 do
3      if x ∈ QRp
4          then d ← d||0
5      else d ← d||1
6          x ← xg-1
7      x ← Sqrt(p, g, x)
8  return d

```

- b. Let $\{u, v\} = \text{Sqrt}(y)$. Then $\text{Lsb}_{t-1}(\text{Log}_{p,g}(u)) \neq \text{Lsb}_{t-1}(\text{Log}_{p,g}(v))$ (the logarithms differ by $\frac{p-1}{2}$). Observe that you can compute these bits by a).

Algorithm 7.3.

```

int A(int p, g, y)
1  {u, v} ← Sqrt(y)
2  if A1(p, g, y) = Lsbt-1(Logp,g(u))
3     then return u
4     else return v

```

A computes the principal square root of y . The assertion now follows by Proposition 7.5.

- c. Let P be a positive polynomial and A_1 be a probabilistic polynomial algorithm, such that

$$\text{prob}(A_1(p, g, g^x) = \text{Lsb}_t(x) : x \xleftarrow{u} \mathbb{Z}_{p-1}) \geq \frac{1}{2} + \frac{1}{P(k)},$$

where p is an odd prime, $p = 2^t a$, a odd, and g is a primitive root mod p . As in b) we get a probabilistic polynomial algorithm A , such that

$$\text{prob}(A(p, g, y) = \text{PSqrt}_{p,g}(y) : y \xleftarrow{u} QR_n) \geq \frac{1}{2} + \frac{1}{P(k)}.$$

This contradicts the discrete logarithm assumption (see Theorem 7.7).

4. a. Let $p-1 = 2^t q$, q odd, $y = g^x$. Compute the t least-significant bits of x by the Algorithm of Exercise 7.2. Guess the next $j-t$ bits from x (note there are only polynomially many, namely $O(k)$, alternatives). Thus, we can assume that the j least-significant bits of x are known.

Algorithm 7.4.

```

int  $A(\text{int } p, g, y)$ 
1   $L_j \leftarrow j$  least-significant bits of  $x$ 
2   $d \leftarrow L_j$ 
3  for  $c \leftarrow j$  to  $k - 1$  do
4       $b \leftarrow A_1(p, g, y, L_j)$ 
5      if  $\text{Lsb}(L_j) = 1$ 
6          then  $y \leftarrow yg^{-1}$ 
7           $\{u, v\} \leftarrow \text{Sqrt}(p, g, y)$ 
8      if  $\text{Lsb}(\text{Log}_{p,g}(u)) = \text{Lsb}_1(L_j)$ 
9          then  $y \leftarrow u$ 
10         else  $y \leftarrow v$ 
11      $L_j \leftarrow b \parallel \text{Lsb}_{j-1}(L_j) \dots \text{Lsb}_1(L_j)$ 
12      $d \leftarrow b \parallel d$ 
13     return  $d$ 

```

- b. The probability of success of A_1 can be increased as in Lemma 7.8. Observe that you can compute $\text{Lsb}_j(x)$ from $\text{Lsb}_j(x+r)$, where r is randomly chosen, if $x+r \leq p-1$. Use this, to compute $\text{Lsb}_j(x)$ with probability almost 1 for small values of x . Then continue as in the proof of Theorem 7.7 to prove statement b).
5. Assume there is a positive polynomial $P \in \mathbb{Z}[X]$ and an algorithm A_1 , such that

$$\begin{aligned} & \text{prob}(A_1(p, g, \text{Lsb}_t(x), \dots, \text{Lsb}_{t+j-1}(x), g^x) \\ & = \text{Lsb}_{t+j}(x) : (p, g) \stackrel{u}{\leftarrow} I_k, x \stackrel{u}{\leftarrow} \mathbb{Z}_{p-1}) > \frac{1}{2} + \frac{1}{P(k)}. \end{aligned}$$

for infinitely many k . By Proposition 6.17, there are polynomials Q, R , such that

$$\begin{aligned} & \text{prob} \left(\left\{ (p, g) \in I_k \mid \text{prob}(A_1(p, g, \text{Lsb}_t(x), \dots, \text{Lsb}_{t+j-1}(x), g^x) \right. \right. \\ & \quad \left. \left. = \text{Lsb}_{t+j}(x) : x \stackrel{u}{\leftarrow} \mathbb{Z}_{p-1}) > \frac{1}{2} + \frac{1}{Q(k)} \right\} \right) > \frac{1}{R(k)}, \end{aligned}$$

for infinitely many k . From the preceding Exercise 4, we conclude that there is an algorithm A_2 and a positive polynomial $S \in \mathbb{Z}[X]$, such that

$$\begin{aligned} & \text{prob} \left(\left\{ (p, g) \in I_k \mid \right. \right. \\ & \quad \left. \left. \text{prob}(A_2(p, g, g^x) = x : x \stackrel{u}{\leftarrow} \mathbb{Z}_{p-1}) \geq 1 - \frac{1}{S(k)} \right\} \right) > \frac{1}{R(k)}, \end{aligned}$$

for infinitely many k . By Proposition 6.3, there is a positive polynomial $T \in \mathbb{Z}[X]$, such that

$$\text{prob}(A_2(p, g, g^x) = x : (p, g) \xleftarrow{u} I_k, x \xleftarrow{u} \mathbb{Z}_{p-1}) > \frac{1}{T(k)},$$

for infinitely many k , a contradiction to the discrete logarithm assumption.

6.

t	a_t	u_t	$a_t x$	Lsb($a_t x$)
0	1	0	13	1
1	15	0.5	21	1
2	22	0.75	25	1
3	11	0.875	27	1
4	20	0.9375	28	0
5	10	0.46875	14	0
6	5	0.234375	7	1

Thus we have $a = 5, u = \frac{15}{64}$.

7.

t	a_t	u_t	returned bits
0	1	0	0
1	196	0	0
2	98	0	1
3	49	0.5	0
4	220	0.25	0
5	110	0.125	1
6	55	0.5625	1
7	223	0.78125	1
8	307	0.890625	1
9	349	0.9453125	0
10	370	0.47265625	1

We get $a = 370, ax = \lfloor \frac{121}{256} 391 + 1 \rfloor$ and $x = a^{-1}ax = 196$.

8. Observe that

$$\begin{aligned} \text{Msb}(x) &= \text{Lsb}(2x) \text{ and} \\ \text{Lsb}(x) &= \text{Msb}(2^{-1}x). \end{aligned}$$

Thus, an algorithm $A(n, e, y)$ computing $\text{Lsb}(x)$ can be used to compute $\text{Msb}(x)$ ($\text{Msb}(x) = A(n, e, 2^e y)$) and vice versa.

9. Follows immediately by Exercise 8.
 10. Let $y = x^e$. Observe that $2^e y = (2x)^e$.

11. **Algorithm 7.5.**

```

int  $RSA^{-1}$ (int  $y$ )
1  for  $i \leftarrow 1$  to  $k - 1$  do
2      if  $LsbRSA^{-1}(y) = 0$ 
3          then  $LSB[i] \leftarrow 0$ 
4               $y \leftarrow y2^{-e} \bmod n$ 
5          else  $LSB[i] \leftarrow 1$ 
6               $y \leftarrow (n - y)2^{-e} \bmod n$ 
7   $t[k] \leftarrow LsbRSA^{-1}(y); t[1] = \dots = t[k - 1] = 0$ 
8  for  $i \leftarrow k - 1$  downto 1 do
9       $t \leftarrow Shift(t)$ 
10     if  $LSB[i] = 1$ 
11         then  $t \leftarrow Delta(n, t, k - i + 1)$ 
12  return  $t$ 
    
```

$Shift(t)$ returns the bits of t , shifted one position to the left, filling the emptied bit with 0. $Delta(s, t, i)$ returns for $t \leq s$ the i least-significant bits of $s - t$. The remaining bits are 0.

12. a. We define

$$L_j : \mathbb{Z}_n^* \longrightarrow \{0, 1\}^j, \quad x \longmapsto x \bmod 2^j.$$

We get the RSA-inversion by rational approximation by using the equations

$$\begin{aligned}
 a_0 &= 1, & u_0 &= 0, \\
 a_t &= 2^{-1}a_{t-1}, & u_t &= \frac{1}{2}(u_{t-1} + Lsb(a_{t-1}x)).
 \end{aligned}$$

We have

$$L_{j-1}(\overline{a_t x}) = \frac{1}{2}L_j(\overline{a_{t-1} x} + Lsb(\overline{a_{t-1} x})n),$$

and we compute $L_j(\overline{a_t x})$ for $t \geq 0$ by

$$\begin{aligned}
 &\text{Guess } L_j(\overline{a_0 x}), \\
 L_j(\overline{a_t x}) &= Lsb_j(\overline{a_t x})2^{j-1} + \frac{1}{2}L_j(\overline{a_{t-1} x} + Lsb(\overline{a_{t-1} x})n).
 \end{aligned}$$

and get

Algorithm 7.6.

```

int  $A_2$ (int  $n, e, y$ )
1   $a \leftarrow 1, u \leftarrow 0$ 
2  guess  $Lst_j \leftarrow L_j(a_0x)$ 
3  for  $t \leftarrow 1$  to  $k$  do
4       $u \leftarrow \frac{1}{2}(u + \text{Lsb}(Lst_j))$ 
5       $a \leftarrow 2^{-1}a \bmod n$ 
6       $Lst_j \leftarrow A_1(n, e, a^e y \bmod n)2^{j-1} + \frac{1}{2}(Lst_j + \text{Lsb}(a_{t-1}x)n)$ 
7  return  $a^{-1} \lfloor un + 1 \rfloor \bmod n$ 

```

b. With the notations from the proof of Theorem 7.14 we have

$$A_{t,i} = a_t + ia_{t-1} + b = (1 + 2i)a_t + b,$$

$$W_{t,i} = \lfloor u_t + iu_{t-1} + v \rfloor.$$

and if $W_{t,i} = q$ (see proof of proof of Theorem 7.14) we have

$$\overline{A_{t,i}x} = \overline{a_t x} + i\overline{a_{t-1}x} + \overline{bx} - W_{t,i}n.$$

Thus, we get

$$L_j(\overline{A_{t,i}x}) = L_j(\overline{a_t x}) + L_j(i\overline{a_{t-1}x}) + L_j(\overline{bx}) - L_j(W_{t,i}n) \bmod 2^j$$

and

$$\text{Lsb}_j(\overline{A_{t,i}x})2^{j-1} + L_{j-1}(\overline{A_{t,i}x}) =$$

$$\text{Lsb}_j(\overline{a_t x})2^{j-1} + L_{j-1}(\overline{a_t x}) + L_j(i\overline{a_{t-1}x}) + L_j(\overline{bx}) -$$

$$L_j(W_{t,i}n) \bmod 2^j, \text{ hence}$$

$$\text{Lsb}_j(\overline{a_t x})2^{j-1} =$$

$$\text{Lsb}_j(\overline{A_{t,i}x})2^{j-1} + L_{j-1}(\overline{A_{t,i}x}) - L_{j-1}(\overline{a_t x} + L_j(i\overline{a_{t-1}x}) - L_j(\overline{bx}) +$$

$$L_j(W_{t,i}n) \bmod 2^j.$$

We use the last equation to get $\text{Lsb}_j(\overline{a_t x})$ by a majority decision computing $\text{Lsb}_j(\overline{A_{t,i}x})$ by algorithm A_1 . Observe that the other terms of the right side of the equation are known. $L_{j-1}(\overline{a_t x})$ and $L_{j-1}(\overline{A_{t,i}x})$ can be recursively computed from $L_j(\overline{a_{t-1}x})$ and $L_j(\overline{A_{t-1,i}x})$:

$$L_{j-1}(\overline{a_t x}) = \frac{1}{2}(L_j(\overline{a_{t-1}x} + \text{Lsb}(\overline{a_{t-1}x})n),$$

$$L_{j-1}(\overline{A_{t,i}x}) = (1 + 2i)L_{j-1}(\overline{a_t x}) + L_{j-1}(\overline{bx}) \bmod 2^{j-1}.$$

Initially we have to guess $L_j(\overline{a_0 x})$ and $L_j(\overline{bx})$. This is polynomial in k , because $j \leq \lfloor \log_2(2k) \rfloor$.

We can modify the Algorithm from Lemma 7.15 to get an algorithm which computes $L_j(a_t x)$ with probability almost 1. From $L_j(a_t x)$ we can easily derive $\text{Lsb}(a_t x)$, and we can use $\text{Lsb}(a_t x)$ in Algorithm 7.17 and continue as in Section 7.2.

13. The proof is analogous to the proof of Exercise 5.

8. One-Way Functions and Pseudorandomness

1. If $\tilde{A}(i, z)$ is a probabilistic polynomial algorithm that distinguishes between the sequences generated by $\pi \circ G$ and true random sequences (see Definition 8.2), then $A(i, z) : (i, z) \mapsto \tilde{A}(i, \Pi(i, z))$ distinguishes the sequences generated by G from true random sequences.
2. Examples can be constructed by one-way permutations $f = (f_i : D_i \rightarrow D_i)_{i \in I}$ with hard-core predicate B , like the RSA family. Consider the pseudorandom generator G with $G_i(x) := (f_i(x), B_i(x))$, which generates from a randomly chosen seed $x \xleftarrow{u} D_i$ a pseudorandom sequence of length $|x| + 1$. G is computationally perfect, by Exercise 7 in Chapter 6. Let π be the permutation $\pi_i(y, b) := (f_i^{-1}(y), b)$ ($y \in D_i, b \in \{0, 1\}$). Then $\pi_i(G_i(x)) = (x, B(x))$, and we see that $\pi \circ G$ is not computationally perfect (since $B(x)$ is computable from x).
3. The proof is an immediate consequence of Exercise 1 (consider the permutation $(x_1, \dots, x_{l(k)}) \mapsto (x_{l(k)}, \dots, x_1)$ and Yao's Theorem 8.7).
4. Assume there is a probabilistic polynomial statistical test $A(i, z)$ and a positive polynomial R , such that

$$\begin{aligned} \text{prob}(A(i, G_i^l(x)) = 1 : i \leftarrow K(1^k), x \xleftarrow{u} \{0, 1\}^{Q(k)}) \\ - \text{prob}(A(i, z) = 1 : i \leftarrow K(1^k), z \xleftarrow{u} \{0, 1\}^{Q(k)+l}) > \frac{1}{R(k)}, \end{aligned}$$

for k in an infinite subset \mathcal{K} of \mathbb{N} (replacing A by $1 - A$ if necessary we may drop the absolute value).

For $k \in \mathcal{K}$ and $i \in I_k$ we consider the following sequence of distributions

$$d_{i,0}, d_{i,1}, \dots, d_{i,l}$$

on $\{0, 1\}^m$, where $m = m(k) := Q(k) + l$.

$$\begin{aligned} d_{i,0} &= \{(b_1, \dots, b_l, x) : (b_1, \dots, b_l) \xleftarrow{u} \{0, 1\}^l, x \xleftarrow{u} \{0, 1\}^{Q(k)}\} \\ d_{i,1} &= \{(b_1, \dots, b_{l-1}, G_i^1(x)) : (b_1, \dots, b_{l-1}) \xleftarrow{u} \{0, 1\}^{l-1}, x \xleftarrow{u} \{0, 1\}^{Q(k)}\} \\ d_{i,2} &= \{(b_1, \dots, b_{l-2}, G_i^2(x)) : (b_1, \dots, b_{l-2}) \xleftarrow{u} \{0, 1\}^{l-2}, x \xleftarrow{u} \{0, 1\}^{Q(k)}\} \\ &\vdots \\ d_{i,r} &= \{(b_1, \dots, b_{l-r}, G_i^r(x)) : (b_1, \dots, b_{l-r}) \xleftarrow{u} \{0, 1\}^{l-r}, x \xleftarrow{u} \{0, 1\}^{Q(k)}\} \\ &\vdots \\ d_{i,l} &= \{G_i^l(x) : x \xleftarrow{u} \{0, 1\}^{Q(k)}\}. \end{aligned}$$

$d_{i,0}$ is the uniform distribution, $d_{i,l}$ is the distribution induced by G_i^l . For $k \in \mathcal{K}$, we have

$$\begin{aligned}
\frac{1}{R(k)} &< \text{prob}(A(i, z) = 1 : i \leftarrow K(1^k), z \stackrel{d_{i,l}}{\leftarrow} \{0, 1\}^m) \\
&\quad - \text{prob}(A(i, z) = 1 : i \leftarrow K(1^k), z \stackrel{d_{i,0}}{\leftarrow} \{0, 1\}^m) \\
&= \sum_{r=0}^{l-1} (\text{prob}(A(i, z) = 1 : i \leftarrow K(1^k), z \stackrel{d_{i,r+1}}{\leftarrow} \{0, 1\}^m) \\
&\quad - \text{prob}(A(i, z) = 1 : i \leftarrow K(1^k), z \stackrel{d_{i,r}}{\leftarrow} \{0, 1\}^m)).
\end{aligned}$$

Define the algorithm \tilde{A} as follows:

- a. Randomly choose r , with $0 \leq r < l$.
- b. Choose random bits $b_1, b_2, \dots, b_{l-r-1}$.
- c. For $z = (z_1, \dots, z_{Q(k)+1}) \in \{0, 1\}^{Q(k)+1}$ let

$$\tilde{A}(i, z) := A(i, b_1, \dots, b_{l-r-1}, z_1, G_i^r((z_2, \dots, z_{Q(k)+1}))).$$

We have

$$\begin{aligned}
&\text{prob}(\tilde{A}(i, G_i(x)) = 1 : i \leftarrow K(1^k), x \stackrel{u}{\leftarrow} \{0, 1\}^{Q(k)}) \\
&\quad - \text{prob}(\tilde{A}(i, z) = 1 : i \leftarrow K(1^k), z \stackrel{u}{\leftarrow} \{0, 1\}^{Q(k)+1}) \\
&= \sum_{r=0}^{l-1} \text{prob}(r) \cdot (\text{prob}(A(i, z) = 1 : i \leftarrow K(1^k), z \stackrel{d_{i,r+1}}{\leftarrow} \{0, 1\}^m) \\
&\quad - \text{prob}(A(i, z) = 1 : i \leftarrow K(1^k), z \stackrel{d_{i,r}}{\leftarrow} \{0, 1\}^m)) \\
&= \frac{1}{l} \sum_{r=0}^{l-1} (\text{prob}(A(i, z) = 1 : i \leftarrow K(1^k), z \stackrel{d_{i,r+1}}{\leftarrow} \{0, 1\}^m) \\
&\quad - \text{prob}(A(i, z) = 1 : i \leftarrow K(1^k), z \stackrel{d_{i,r}}{\leftarrow} \{0, 1\}^m)) \\
&> \frac{1}{lR(k)},
\end{aligned}$$

for the infinitely many $k \in \mathcal{K}$. This contradicts the assumption that G is computationally perfect.

5. The proof runs in the same way as the proof of Yao's Theorem 8.7. An additional input $y \in Y_i$ has to be added to the algorithms A and \tilde{A} and the probabilities

$$\text{prob}(\tilde{A}(i, f_i(x), z) = \dots : i \leftarrow K(1^k), x \stackrel{u}{\leftarrow} X_i, z \leftarrow \dots)$$

must also be taken over $x \stackrel{u}{\leftarrow} X_i$. The distributions $p_{i,r}$ are modified to

$$\begin{aligned}
p_{i,r} = \{ &(f_i(x), G_{i,1}(x), G_{i,2}(x), \dots, G_{i,r}(x), b_{r+1}, \dots, b_{Q(k)} : \\
&(b_{r+1}, \dots, b_{Q(k)}) \stackrel{u}{\leftarrow} \{0, 1\}^{Q(k)-r}, x \stackrel{u}{\leftarrow} X_i \}.
\end{aligned}$$

6. Assume there is a probabilistic polynomial algorithm $A(i, y)$, such that

$$\begin{aligned} \text{prob}(A(i, f_i(x)) = C_i(B_{i,1}(x), \dots, B_{i,l(k)}(x)) : i \leftarrow K(1^k), x \stackrel{u}{\leftarrow} D_i) \\ > \frac{1}{2} + \frac{1}{P(k)}, \end{aligned}$$

for k in an infinite subset \mathcal{K} of \mathbb{N} .

Define the algorithm $\tilde{A}(i, y, z_1, \dots, z_l)$ as follows:

$$\tilde{A}(i, y, z_1, \dots, z_l) := \begin{cases} 1 & \text{if } A(i, y) = C_i(z_1, \dots, z_l), \\ 0 & \text{else} \end{cases}$$

We have

$$\begin{aligned} \text{prob}(A(i, f_i(x)) = C_i(z_1, \dots, z_l) : i \leftarrow K(1^k), x \stackrel{u}{\leftarrow} D_i, \\ (z_1, \dots, z_l) \stackrel{u}{\leftarrow} \{0, 1\}^l) \\ = \text{prob}(A(i, f_i(x)) = 0 : i \leftarrow K(1^k), x \stackrel{u}{\leftarrow} D_i) \\ \quad \cdot \text{prob}(C_i(z_1, \dots, z_l) = 0 : (z_1, \dots, z_l) \stackrel{u}{\leftarrow} \{0, 1\}^l) \\ \quad + \text{prob}(A(i, f_i(x)) = 1 : i \leftarrow K(1^k), x \stackrel{u}{\leftarrow} D_i) \\ \quad \cdot \text{prob}(C_i(z_1, \dots, z_l) = 1 : (z_1, \dots, z_l) \stackrel{u}{\leftarrow} \{0, 1\}^l) \\ = \text{prob}(A(i, f_i(x)) = 0 : i \leftarrow K(1^k), x \stackrel{u}{\leftarrow} D_i) \cdot \frac{1}{2} \\ \quad + \text{prob}(A(i, f_i(x)) = 1 : i \leftarrow K(1^k), x \stackrel{u}{\leftarrow} D_i) \cdot \frac{1}{2} \\ = \frac{1}{2}. \end{aligned}$$

Hence

$$\begin{aligned} & \left| \text{prob}(\tilde{A}(i, f_i(x), z_1, \dots, z_l) = 1 : i \leftarrow K(1^k), x \stackrel{u}{\leftarrow} D_i, (z_1, \dots, z_l) \stackrel{u}{\leftarrow} \{0, 1\}^l) \right. \\ & \quad \left. - \text{prob}(\tilde{A}(i, f_i(x), B_{i,1}(x), \dots, B_{i,l}(x)) = 1 : i \leftarrow K(1^k), x \stackrel{u}{\leftarrow} D_i) \right| \\ & = \left| \frac{1}{2} - \text{prob}(A(i, f_i(x)) = C_i(B_{i,1}(x), \dots, B_{i,l}(x))) : i \leftarrow K(1^k), x \stackrel{u}{\leftarrow} D_i \right| \\ & > \frac{1}{2} + \frac{1}{P(k)} - \frac{1}{2} \\ & > \frac{1}{P(k)}, \end{aligned}$$

for infinitely many k . This is a contradiction.

7. Assume that the bits $B_{i,1}, \dots, B_{i,l}$ are not simultaneously secure. From the stronger version of Yao's Theorem, Exercise 5, we conclude that there is a probabilistic polynomial algorithm A , a positive polynomial P and a $j_k, 1 \leq j_k \leq l(k)$, such that

$$\begin{aligned} \text{prob}(A(i, f_i(x), B_{i,1}(x) \dots B_{i,j_k-1}(x)) = B_{i,j_k}(x) : i \leftarrow K(1^k), x \stackrel{u}{\leftarrow} X_i) \\ > \frac{1}{2} + \frac{1}{P(k)}, \end{aligned}$$

for infinitely many k . This is a contradiction.

8. The statement, which is analogous to Theorem 8.4, is almost identical to the statement of Theorem 8.4:

For every probabilistic polynomial algorithm A with inputs $i \in I_k, z \in \{0, 1\}^{l(k)Q(k)}, y \in D_i$ and output in $\{0, 1\}$ and every positive polynomial $P \in \mathbb{Z}[X]$, there is a $k_0 \in \mathbb{N}$, such that for all $k \geq k_0$

$$\begin{aligned} |\text{prob}(A(i, z, y) = 1 : i \leftarrow K(1^k), z \stackrel{u}{\leftarrow} \{0, 1\}^{l(k)Q(k)}, y \stackrel{u}{\leftarrow} D_i) \\ - \text{prob}(A(i, G_i(x), f_i^{Q(k)}(x)) = 1 : i \leftarrow K(1^k), x \stackrel{u}{\leftarrow} D_i) | \leq \frac{1}{P(k)}. \end{aligned}$$

The proof runs as the proof of Theorem 8.4. There are only the following differences:

In the distributions $p_{i,r}$, the elements b_i have to be chosen from $\{0, 1\}^{l(k)}$: $b_i \stackrel{u}{\leftarrow} \{0, 1\}^{l(k)}$, and X_i has to be set as $X_i := \{0, 1\}^{l(k)Q(k)} \times D_i$.

We define the algorithm \tilde{A} as follows:

On inputs $i \in I_k, y \in D_i, w \in \{0, 1\}^{l(k)}$

- a. Randomly choose r , with $0 \leq r < Q(k)$.
- b. Randomly choose $b_1, b_2, \dots, b_{Q(k)-r-1}$ in $\{0, 1\}^{l(k)}$.
- c. For $y = f_i(x)$ let $\tilde{A}(i, y, w) :=$

$$A(i, b_1, \dots, b_{Q(k)-r-1}, w, B_i(f_i(x)), B_i(f_i^2(x)), \dots, B_i(f_i^r(x)), f_i^{r+1}(x)).$$

Then

$$\begin{aligned} |\text{prob}(\tilde{A}(i, f_i(x), B_i(x)) = 1 : i \leftarrow K(1^k), x \stackrel{u}{\leftarrow} D_i) \\ - \tilde{A}(i, y, w) = 1 : i \leftarrow K(1^k), y \stackrel{u}{\leftarrow} D_i, w \stackrel{u}{\leftarrow} \{0, 1\}^{l(k)} | \\ = \sum_{r=0}^{Q(k)-1} \text{prob}(r) \cdot (\text{prob}(A(i, z, y) = 1 : i \leftarrow K(1^k), (z, y) \stackrel{p_{i,r+1}}{\leftarrow} X_i) \\ - \text{prob}(A(i, z, y) = 1 : i \leftarrow K(1^k), (z, y) \stackrel{p_{i,r}}{\leftarrow} X_i)) \\ = \frac{1}{l(k)} \sum_{r=0}^{Q(k)-1} (\text{prob}(A(i, z, y) = 1 : i \leftarrow K(1^k), (z, y) \stackrel{p_{i,r+1}}{\leftarrow} X_i) \\ - \text{prob}(A(i, z, y) = 1 : i \leftarrow K(1^k), (z, y) \stackrel{p_{i,r}}{\leftarrow} X_i)) \\ > \frac{1}{l(k)P(k)}, \end{aligned}$$

for infinitely many k . This contradicts the fact that B is an l -bit hard-core predicate.

9. Provably Secure Encryption

1. The affine cipher is not perfectly secret.

Namely, let $m \in \mathbb{Z}_n, r := \gcd(m, n), m = rm'$ and $c \in \mathbb{Z}_n$. We look for the number of keys (a, b) , such that m is encrypted as c . If the affine cipher were perfectly secret, then this number would be the same for all plaintexts m .

We have $c = a \cdot m + b = a \cdot r \cdot m' + b$ if and only if $c - b \equiv 0 \pmod r$ and $a = ((c - b)/r) \cdot m'^{-1}$ (modulo n). Hence, there are solutions of $c = a \cdot m + b$ for n/r distinct values of b . Fix such a b . Then, $c = a \cdot m + b$ has r distinct solutions for a in \mathbb{Z}_n : If a is a solution, then all solutions are $a_i := a + i \cdot n/r \pmod n, i = 0, 1, \dots, r - 1$. If $r = 1$ (i.e., m is a unit modulo n), then the number of keys (a, b) is n . However, in general, if m is not a unit, then not all the solutions a_i are units in \mathbb{Z}_n^* and hence, the number of keys is $< n$. For example, take $n = 3 \cdot 5 = 15$ and $m = 9, c = 5$. Then, the number of keys is $8 < 15$.

2. Knowing the key and the ciphertext, the plaintext m can be derived. Hence, $H(M|KC) = 0$. Therefore, we have

$$\begin{aligned} 0 \leq I(M; K|C) &= H(K|C) - H(K|MC) \\ &= H(M|C) - H(M|KC) = H(M|C). \end{aligned}$$

Hence, $H(K|C) \geq H(M|C)$, because $H(K|MC) \geq 0$.

3. It is not computationally secret, as the following considerations show. Let $(p, g, y := g^x)$ be an ElGamal public key. We have $E_{p,g,y}(m) = (g^k, y^k m)$ for plaintexts $m \in \mathbb{Z}_p^*$. Applying $\text{Log}_{p,g}$ to both components of $E_{p,g,y}(m)$, we get $(k, kx + \text{Log}_{p,g}(m))$. If $p = 2^t a, a$ odd, then the t least-significant bits of $\text{Log}_{p,g}(z)$ can be easily computed for $z \in \mathbb{Z}_p^*$ (see Section 7.1, Exercise 3 in Chapter 7). In particular, we can compute the t least-significant bits of $k, x, \text{Log}_{p,g}(y^k m)$. Since $(kx \pmod{p-1}) \pmod{2^t} = kx \pmod{2^t}$, we can compute the t least-significant bits of kx and hence also of $\text{Log}_{p,g}(m)$. Thus, we can distinguish between plaintexts, whose $\text{Log}_{p,g}$ differ in their t least-significant bits.

If we consider only the least-significant bit, this means that we can distinguish between quadratic residues and non-residues.

4. Note that $S(i)$ returns two distinct messages $m_0 \neq m_1$. We have for every pair $m_0 \neq m_1$

$$\begin{aligned} \text{prob}(A(i, m_0, m_1, c) = m : i \leftarrow K(1^k), m \stackrel{u}{\leftarrow} \{m_0, m_1\}, c \leftarrow E(m)) \\ &= \frac{1}{2} \cdot \text{prob}(A(n, e, m_0, m_1, c) = m_0 : (n, e) \stackrel{u}{\leftarrow} I_k, c \leftarrow E(m_0)) \\ &\quad + \frac{1}{2} \cdot \text{prob}(A(n, e, m_0, m_1, c) = m_1 : (n, e) \stackrel{u}{\leftarrow} I_k, c \leftarrow E(m_1)) \\ &= \frac{1}{2} + \frac{1}{2} \cdot [\text{prob}(A(n, e, m_0, m_1, c) = m_0 : (n, e) \stackrel{u}{\leftarrow} I_k, c \leftarrow E(m_0)) \end{aligned}$$

$$- \text{prob}(A(n, e, m_0, m_1, c) = m_0 : (n, e) \stackrel{u}{\leftarrow} I_k, c \leftarrow E(m_1)).$$

5. For $m \in \{0, 1\}^r$, we denote by \bar{m} the padded m .

Assume that the encryption scheme is not computationally secret. Then, by Exercise 4, there is a probabilistic polynomial algorithm A and a positive polynomial P , such that for infinitely many k : For all $(n, e) \in I_k$ there are $m_{0,n,e}, m_{1,n,e} \in \{0, 1\}^r, m_{0,n,e} \neq m_{1,n,e}$, such that

$$\begin{aligned} & \text{prob}(A(n, e, m_{0,n,e}, m_{1,n,e}, c) = m_{0,n,e} : (n, e) \stackrel{u}{\leftarrow} I_k, c \leftarrow \text{RSA}_{n,e}(\overline{m_{0,n,e}})) \\ & - \text{prob}(A(n, e, m_{0,n,e}, m_{1,n,e}, c) = m_{0,n,e} : (n, e) \stackrel{u}{\leftarrow} I_k, \\ & \hspace{15em} c \leftarrow \text{RSA}_{n,e}(\overline{m_{1,n,e}})) \\ & > \frac{1}{P(k)}. \end{aligned}$$

Here, observe that there are only polynomially many, namely $< 4k^2$, message pairs $\{m_0, m_1\}$, so we can omit the sampling algorithm S (all message pairs can be considered in polynomial time).

Let Q be a positive polynomial with $\deg(Q) > \deg(P) + 1$. Replacing A by a modification, if necessary, we may assume that the probability of those $(n, e) \stackrel{u}{\leftarrow} I_k, m_0, m_1 \stackrel{u}{\leftarrow} \{0, 1\}^r$, such that either

$$\begin{aligned} & \text{prob}(A(n, e, m_0, m_1, c) = m_0 : c \leftarrow \text{RSA}_{n,e}(\overline{m_0})) \\ & - \text{prob}(A(n, e, m_0, m_1, c) = m_0 : c \leftarrow \text{RSA}_{n,e}(\overline{m_1})) \\ & \geq 0. \end{aligned}$$

or the absolute value of the difference is $\leq 1/Q(k)$, is $\geq 1 - 1/Q(k)$. (The sign of the difference may be computed by a probabilistic polynomial algorithm with high probability, see Proposition 6.18 and, e.g., the proof of Proposition 6.17. Replace the output by its complement, if the sign is negative).

Then

$$\begin{aligned} & \text{prob}(A(n, e, m_0, m_1, c) = m_0 : (n, e) \stackrel{u}{\leftarrow} I_k, m_0 \stackrel{u}{\leftarrow} \{0, 1\}^r, \\ & \hspace{10em} m_1 \stackrel{u}{\leftarrow} \{0, 1\}^r \setminus \{m_0\}, c \leftarrow \text{RSA}_{n,e}(\overline{m_0})) \\ & - \text{prob}(A(n, e, m_0, m_1, c) = m_0 : (n, e) \stackrel{u}{\leftarrow} I_k, m_0 \stackrel{u}{\leftarrow} \{0, 1\}^r, \\ & \hspace{10em} m_1 \stackrel{u}{\leftarrow} \{0, 1\}^r \setminus \{m_0\}, c \leftarrow \text{RSA}_{n,e}(\overline{m_1})) \\ & > \frac{1}{2^{2r}} \frac{1}{2P(k)} \geq \frac{1}{8k^2 P(k)}. \end{aligned}$$

Let \tilde{A} be the following algorithm with inputs $(n, e) \in I, y \in \mathbb{Z}_n, z \in \{0, 1\}^r$:

a. Randomly select $m_1 \stackrel{u}{\leftarrow} \{0, 1\}^r, z \neq m_1$.

$$\text{b. } \tilde{A}(n, e, y, z) := \begin{cases} 1 & \text{if } A(n, e, z, m_1, y) = z, \\ 0 & \text{else.} \end{cases}$$

Then

$$\begin{aligned} \text{prob}(\tilde{A}(n, e, \text{RSA}_{n,e}(\bar{z}), z) = 1 : (n, e) \stackrel{u}{\leftarrow} I_k, z \stackrel{u}{\leftarrow} \{0, 1\}^r) \\ - \text{prob}(\tilde{A}(n, e, y, z) = 1 : (n, e) \stackrel{u}{\leftarrow} I_k, y \stackrel{u}{\leftarrow} \mathbb{Z}_n, z \stackrel{u}{\leftarrow} \{0, 1\}^r) \\ \approx \text{prob}(A(n, e, z, m_1, y) = z : (n, e) \stackrel{u}{\leftarrow} I_k, z \stackrel{u}{\leftarrow} \{0, 1\}^r, \\ m_1 \stackrel{u}{\leftarrow} \{0, 1\}^r \setminus \{z\}, y \leftarrow \text{RSA}_{n,e}(\bar{z})) \\ - \text{prob}(A(n, e, z, m_1, y) = z : (n, e) \stackrel{u}{\leftarrow} I_k, z \stackrel{u}{\leftarrow} \{0, 1\}^r, \\ m_1 \stackrel{u}{\leftarrow} \{0, 1\}^r \setminus \{z\}, y \leftarrow \text{RSA}_{n,e}(\overline{m_1})) \\ > \frac{1}{2^{2r}} \frac{1}{2P(k)} \geq \frac{1}{8k^2 P(k)}, \end{aligned}$$

for infinitely many k .

To justify \approx , observe that $y \stackrel{u}{\leftarrow} \mathbb{Z}_n$ is the same as $m_1 \stackrel{u}{\leftarrow} \{0, 1\}^r, y \leftarrow \text{RSA}_{n,e}(\overline{m_1})$ ($\text{RSA}_{n,e}$ is bijective!), hence polynomially close to $m_1 \stackrel{u}{\leftarrow} \{0, 1\}^r \setminus \{z\}, y \leftarrow \text{RSA}_{n,e}(\overline{m_1})$.

We obtained a contradiction to the fact that the $r \leq \log_2(|n|)$ least significant bits of RSA are simultaneously secure (see Exercise 7 in Chapter 8, there only units are considered as inputs to RSA, but $y \stackrel{u}{\leftarrow} \mathbb{Z}_n$ is polynomially close to $y \stackrel{u}{\leftarrow} \mathbb{Z}_n^*$ (Lemma B.23)).

6. In order to decrypt, the recipient of the encrypted message $c_1 \dots c_n$ uses his secret trapdoor information to compute the elements $x_j = f_i^{-1}(c_j)$. Then, he obtains m as $B_i(x_1) \dots B_i(x_n)$.

To prove security, assume that the scheme is not computationally secret. Let S be a sampling algorithm and A be a distinguishing algorithm, such that

$$\begin{aligned} \text{prob}(A(i, m_0, m_1, c) = m_0 : i \leftarrow K(1^k), \{m_0, m_1\} \leftarrow S(i), c \leftarrow E(m_0)) \\ - \text{prob}(A(i, m_0, m_1, c) = m_1 : i \leftarrow K(1^k), \{m_0, m_1\} \leftarrow S(i), c \leftarrow E(m_0)) \\ > \frac{1}{P(k)}, \end{aligned}$$

for some positive polynomial P and infinitely many k (see Exercise 4).

For $m_0, m_1 \in \{0, 1\}^n$ and $0 \leq r \leq n$, we denote by $s_r(m_0, m_1)$ the concatenation of the first $n - r$ bits of m_0 with the last r bits of m_1 . Thus, $s_0(m_0, m_1) = m_0$ and $s_n(m_0, m_1) = m_1$. We denote by $m_{j,l}$ the l -th bit of m_j . Then $s_r(m_0, m_1) = m_{0,1}m_{0,2} \dots m_{0,n-r}m_{1,n-r+1} \dots m_{1,n}$. For $0 \leq r \leq n$, let

$$\begin{aligned} p_r := \text{prob}(A(i, m_0, m_1, c) = m_1 : i \stackrel{u}{\leftarrow} K(1^k), \\ \{m_0, m_1\} \leftarrow S(i), c \leftarrow E(i, s_r(m_0, m_1))). \end{aligned}$$

and

$$\begin{aligned}
 & p_{r, m_{0,l}=m_{1,l}} \\
 & := \text{prob}(A(i, m_0, m_1, c) = m_1 \mid m_{0,l} = m_{1,l} : \\
 & \quad i \stackrel{u}{\leftarrow} k(1^k), \{m_0, m_1\} \leftarrow S(i), c \leftarrow E(i, s_r(m_0, m_1))).
 \end{aligned}$$

be the conditional probability assuming that $m_{0,l} = m_{1,l}$. Analogously for the condition $m_{0,l} \neq m_{1,l}$.

With this notation, we have $p_n - p_0 > \frac{1}{P(k)}$. Since $p_n - p_0 = \sum_{r=0}^n (p_{r+1} - p_r)$, there is some $r, 0 \leq r \leq n$, with $p_{r+1} - p_r > \frac{1}{nP(k)}$ (Recall $n = Q(k)$). $s_r(m_0, m_1)$ and $s_{r+1}(m_0, m_1)$ differ only in the $l = n - r - 1$ -th bit. Hence $s_r(m_0, m_1) = s_{r+1}(m_0, m_1)$, if $m_{0,l} = m_{1,l}$, and thus $p_{r, m_{0,l}=m_{1,l}} = p_{r+1, m_{0,l}=m_{1,l}}$. Therefore, the inequality $p_{r+1} - p_r > \frac{1}{nP(k)}$ also implies

$$\text{prob}(m_{0,l} \neq m_{1,l}) \cdot (p_{r+1, m_{0,l} \neq m_{1,l}} - p_{r, m_{0,l} \neq m_{1,l}}) > \frac{1}{nP(k)}.$$

We can approximately compute the probabilities p_r by a probabilistic polynomial algorithm, with high probability (Proposition 6.18). We conclude that for a given positive polynomial T , there is a probabilistic polynomial algorithm that on input 1^k computes an r with $p_{r+1} - p_r > 1/nP(k)$, with probability $\geq 1 - 1/T(k)$.

Now, we give an algorithm $\tilde{A}(i, y)$ which successfully computes the predicate B . In a preprocessing phase, \tilde{A} computes an r with $p_{r+1} - p_r > 1/nP(k)$ (with probability $\geq 1 - 1/T(k)$). \tilde{A} then uses this r for all inputs (i, y) with $i \in I_k$. Let $l := n - r - 1$. Note that $s_r(m_0, m_1)$ and $s_{r+1}(m_0, m_1)$ differ only in the l -th bit. On input (i, y) , \tilde{A} works as follows:

- a. Compute $\{m_0, m_1\} \leftarrow S(i)$.
- b. If $m_{0,l} = m_{1,l}$, then return a random $b \stackrel{u}{\leftarrow} \{0, 1\}$ and stop.
- c. Else, i.e., if $m_{0,l} \neq m_{1,l}$, randomly (and uniformly) choose x_1, \dots, x_n , such that $B_i(x_j)$ equals the j -th bit of $s_r(m_0, m_1)$. Let $y_j := f_i(x_j)$. (Note that $y_1 \| y_2 \| \dots \| y_n$ is an encryption of $s_r(m_0, m_1)$.)
- d. Let $c := y_1 \| \dots \| y_{l-1} \| y \| y_{l+1} \| \dots \| y_n$.
- e. If $A(i, m_0, m_1, c) = m_0$, then return $\tilde{A}(i, y) = B_i(x_l) = m_{0,l}$. Else, return $\tilde{A}(i, y) = 1 - B_i(x_l) = 1 - m_{0,l} = m_{1,l}$.

We want to prove that for some positive polynomial R and infinitely many k ,

$$\begin{aligned}
 & \frac{1}{2} + \frac{1}{R(k)} < \text{prob}(\tilde{A}(i, f_i(x)) = B_i(x) : i \stackrel{u}{\leftarrow} k(1^k), x \stackrel{u}{\leftarrow} D_i) \\
 & = \text{prob}(m_{0,l} = m_{1,l}) \cdot \text{prob}(\tilde{A}(i, f_i(x)) = B_i(x) \mid m_{0,l} = m_{1,l})
 \end{aligned}$$

$$\begin{aligned}
 & + \text{prob}(m_{0,l} \neq m_{1,l}) \cdot \text{prob}(\tilde{A}(i, f_i(x)) = B_i(x) | m_{0,l} \neq m_{1,l}) \\
 = & \text{prob}(m_{0,l} = m_{1,l}) \cdot \frac{1}{2} \\
 & + \text{prob}(m_{0,l} \neq m_{1,l}) \cdot \text{prob}(\tilde{A}(i, f_i(x)) = B_i(x) | m_{0,l} \neq m_{1,l}) \\
 =: & (1),
 \end{aligned}$$

where the probabilities are taken over $i \stackrel{u}{\leftarrow} k(1^k), x \stackrel{u}{\leftarrow} D_i$ and the coin tosses. This will be the desired contradiction to the fact that B is a hard-core predicate. The following probabilities are computed under the assumption that $m_{0,l} \neq m_{1,l}$ (we omit the assumption in our notation).

$$\begin{aligned}
 & \text{prob}(\tilde{A}(i, f_i(x)) = B_i(x)) \\
 = & \text{prob}(B_i(x) = m_{0,l}) \cdot \text{prob}(\tilde{A}(i, f_i(x)) = m_{0,l} | B_i(x) = m_{0,l}) \\
 & + \text{prob}(B_i(x) = m_{1,l}) \cdot \text{prob}(\tilde{A}(i, f_i(x)) = m_{1,l} | B_i(x) = m_{1,l}) \\
 = & \frac{1}{2}q_1 + \frac{1}{2} \cdot q_2 + \varepsilon \\
 =: & (2)
 \end{aligned}$$

with

$$\begin{aligned}
 q_1 := & \text{prob}(A(i, m_0, m_1, c) = m_0 : i \stackrel{u}{\leftarrow} k(1^k), \\
 & \{m_0, m_1\} \leftarrow S(i), c \leftarrow E(i, s_r(m_0, m_1))) \\
 = & 1 - \text{prob}(A(i, m_0, m_1, c) = m_1 : i \stackrel{u}{\leftarrow} k(1^k), \\
 & \{m_0, m_1\} \leftarrow S(i), c \leftarrow E(i, s_r(m_0, m_1))) \\
 = & 1 - p_{r, m_{0,l} \neq m_{1,l}},
 \end{aligned}$$

$$\begin{aligned}
 q_2 := & \text{prob}(A(i, m_0, m_1, c) = m_1 : i \stackrel{u}{\leftarrow} k(1^k), \\
 & \{m_0, m_1\} \leftarrow S(i), c \leftarrow E(i, s_{r+1}(m_0, m_1))) \\
 = & p_{r+1, m_{0,l} \neq m_{1,l}}
 \end{aligned}$$

and a negligibly small ε , i.e., given a positive polynomial U , $\varepsilon \leq 1/U(k)$ for sufficiently large k (see Exercise 7 in Chapter 6).

Thus

$$(2) = \frac{1}{2} + p_{r+1, m_{0,l} \neq m_{1,l}} - p_{r, m_{0,l} \neq m_{1,l}} + \varepsilon.$$

We insert (2) in (1) and get

$$\begin{aligned}
 (1) = & \frac{1}{2} + \text{prob}(m_{0,l} \neq m_{1,l}) \cdot (p_{r+1, m_{0,l} \neq m_{1,l}} - p_{r, m_{0,l} \neq m_{1,l}} + \varepsilon) \\
 > & \frac{1}{2} + (1 - \frac{1}{T(k)}) \cdot \frac{1}{nP(k)} + \varepsilon \\
 > & \frac{1}{2} + \frac{1}{2nP(k)} = \frac{1}{2} + \frac{1}{2Q(k)P(k)},
 \end{aligned}$$

for infinitely many k .
 The proof is finished.

7. To decrypt an encrypted message $c_1 \dots c_n$, Bob checks (by using the factorization of n), whether c_j is a quadratic residue or not. The security proof is almost identical to the proof of Exercise 6. It leads to a contradiction to statement 2 in Exercise 9 in Chapter 6, which is equivalent to the quadratic residuosity assumption (see Exercise 9).
8. a) Let $x_0, x_1 \in \mathbb{F}_{2^l}, x_0 \neq x_1$. Multiplying in \mathbb{F}_{2^l} by some element $x \in \mathbb{F}_{2^l}$ is a linear map over \mathbb{F}_2 . Thus, $a \mapsto a \cdot (x_1 - x_0)$ can be computed by an $l \times l$ -matrix M over \mathbb{F}_2 . M is invertible, because $x_1 \neq x_0$. Let M' be the first f rows of M . Then M' has rank f . Therefore

$$|\{a \in \mathbb{F}_{2^l} \mid M' \cdot a = 0\}| = 2^{l-f} \text{ and hence}$$

$$\text{prob}(\text{msb}(a \cdot x_0) = \text{msb}(a \cdot x_1) : a \xleftarrow{u} \mathbb{F}_{2^l}) = 2^{l-f} \cdot 2^{-l} = 2^{-f}.$$

- b) Let $x_0, x_1, z_0, z_1 \in \mathbb{F}_{2^l}, x_0 \neq x_1$ and $y_0, y_1 \in \mathbb{F}_{2^f}$. The equation

$$\begin{pmatrix} x_0 & 1 \\ x_1 & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \begin{pmatrix} z_0 \\ z_1 \end{pmatrix}$$

has exactly one solution, since $x_0 \neq x_1$ and hence, the matrix is invertible. Thus

$$|\{(a_0, a_1) \mid h_{a_0, a_1}(x_0) = y_0, h_{a_0, a_1}(x_1) = y_1\}| = 2^{l-f} 2^{l-f} \text{ and}$$

$$\begin{aligned} \text{prob}(h_{a_0, a_1}(x_0) = y_0, h_{a_0, a_1}(x_1) = y_1 : a_0 \xleftarrow{u} \mathbb{F}_{2^l}, a_1 \xleftarrow{u} \mathbb{F}_{2^l}) &= 2^{-f} \cdot 2^{-f} \\ &= \frac{1}{|\mathbb{F}_{2^f}|^2}. \end{aligned}$$

10. Provably Secure Digital Signatures

1. We can use a pair of claw-free one-way permutations to construct a collision-resistant compression function $\{0, 1\}^{l(k)} \rightarrow \{0, 1\}^{g(k)}$, $m \mapsto f_{m,i}(x)$ with some $l(k) > g(k)$, as in Section 10.2. The collision resistance can be proven as in the proof of Proposition 10.7. Here, the prefix-free encoding is not necessary, since all strings in the domain have the same binary length. Then, we can derive a provably collision-resistant family of hash functions by applying Merkle's meta method.
2. Let K be the key generator of \mathcal{H} .
Let $A(i, y)$ be a probabilistic algorithm with output in $\{0, 1\}^{\leq l_i}$ which successfully computes pre-images, i.e., there is a positive polynomial P , such that

$$\text{prob}(A(i, h_i(x)) \in h_i^{-1}(h_i(x)) : i \leftarrow K(1^k), x \xleftarrow{u} \{0, 1\}^{\leq l_i}) \geq \frac{1}{P(k)}$$

for k in an infinite subset $\mathcal{K} \subseteq \mathbb{N}$.

By D_k we denote the subset $\{(i, x) \mid i \in I_k, x \in \{0, 1\}^{\leq l_i}, h_i^{-1}(h_i(x)) = \{x\}\}$ of those (i, x) where x is the only pre-image of $h_i(x)$ and by D_i be the set of elements of D_k with key i . h_i maps D_i injectively to $\{0, 1\}^{g(k)}$. Thus D_i contains at most $2^{g(k)}$ elements. Moreover, we have $l_i \geq g(k) + k$ by assumption, thus

$$\text{prob}(D_k) \leq \sum_{i \in I_k} \text{prob}(i) \cdot \frac{2^{g(k)}}{2^{l_i+1} - 1} \leq \sum_{i \in I_k} \text{prob}(i) \frac{1}{2^k} = \frac{1}{2^k}.$$

In the computation, observe that the number of bit strings $\leq l_i$ is equal to $\sum_{j=1}^{l_i} 2^j = 2^{l_i+1} - 1$.

Let $\overline{D}_i := \{0, 1\}^{\leq l_i} \setminus D_i$ be the complement of D_i .

Lemma B.10 tells us that

$$\text{prob}(A(i, h_i(x)) \in h_i^{-1}(h_i(x)) : i \leftarrow K(1^k), x \xleftarrow{u} \overline{D}_i) \geq \frac{1}{P(k)} - \frac{1}{2^k} \geq \frac{1}{2P(k)}$$

for k in an infinite subset $\mathcal{K}' \subseteq \mathbb{N}$.

Now let $\tilde{A}(i)$ be the following algorithm:

- a. Randomly choose $x \xleftarrow{u} \{0, 1\}^{\leq l_i}$.
- b. Return $(x, A(i, h_i(x)))$.

If $x \neq A(i, h_i(x))$ and $A(i, h_i(x)) \in h_i^{-1}(h_i(x))$, then \tilde{A} returns a collision of \mathcal{H} . (In fact, the algorithm computes a second pre-image. Therefore, our proof will even show that second-pre-image resistance implies the one-way property.)

We compute the probability of this event.

$$\begin{aligned}
 & \text{prob}(x \neq A(i, h_i(x)), A(i, h_i(x)) \in h_i^{-1}(h_i(x)) : \\
 & \qquad \qquad \qquad i \leftarrow K(1^k), x \xleftarrow{u} \{0, 1\}^{\leq l_i}) \\
 & \geq \text{prob}(x \neq A(i, h_i(x)), A(i, h_i(x)) \in h_i^{-1}(h_i(x)) : \\
 & \qquad \qquad \qquad i \leftarrow K(1^k), x \xleftarrow{u} \overline{D}_i) - \frac{1}{2^k} \text{ (Lemma B.10)} \\
 & = \text{prob}(A(i, h_i(x)) \in h_i^{-1}(h_i(x)) : i \leftarrow K(1^k), x \xleftarrow{u} \overline{D}_i) \\
 & \quad \cdot \text{prob}(x \neq A(i, h_i(x)) | A(i, h_i(x)) \in h_i^{-1}(h_i(x)) : \\
 & \qquad \qquad \qquad i \leftarrow K(1^k), x \xleftarrow{u} \overline{D}_i) - \frac{1}{2^k} \\
 & \geq \frac{1}{2P(k)} \cdot \frac{1}{2} - \frac{1}{2^k} \\
 & \geq \frac{1}{5P(k)}
 \end{aligned}$$

for infinitely many $k \in \mathcal{K}'$. This is a contradiction, since \mathcal{H} is assumed to be collision-resistant. Note that for $x \in \overline{D}_i$ the fibre $h_i^{-1}(h_i(x))$ contains more than one element. So, for a randomly chosen x , the probability that $x \neq A(i, h_i(x))$ is $\geq 1/2$.

3. a) RSA:

The attacks discussed in Section 3.3.1 are key-only attacks against the RSA one-way function which may result in the retrieval of secret keys. Forging signed messages (m^e, m) (Section 3.3.2) is an existential forgery by a key-only attack. The “homomorphism attacks” can be used for universal forgery by chosen-message attacks.

b) ElGamal:

The retrieval of secret keys is possible, if the random number k is figured out by the adversary in a known-signatures attack (see Section 3.5.2). Existential forgery by a key-only attack is possible (loc.cit.). If step 1 in the verification procedure is omitted, then signatures can be universally forged by a known-signature attack, as Bleichenbacher observed (loc.cit.).

4. a. Retrieving the secret key by a key-only attack means to determine the private key x from $y = -x^{-2}$. Since x is chosen randomly, this means that the adversary has a probabilistic algorithm $A(n, z)$ that computes square roots from randomly chosen elements $z \in QR_n$, with a non-negligible probability (the probability taken over the random choice of n and x). Then, the adversary can also compute the prime factors of n with a non-negligible probability (Proposition A.64).
- b. Take any s_1, s_2 and compute $m := s_1^2 + ys_2^2$. Then, (s_1, s_2) is a valid signature for m .
- c. For a given m , about n of the n^2 pairs (s_1, s_2) are solutions of $m = s_1^2 + ys_2^2$. Choosing a pair randomly (and uniformly), the probability that it is a solution is about $n^{-1} \approx 2^{-|n|}$ and hence negligible.

- d. The adversary has to own a probabilistic polynomial algorithm $A(n, y, m)$ which yields solutions of $m = s_1^2 + ys_2^2$ (modulo n) with a non-negligible probability.
5. a. We have $f_{[m],i}(\sigma(i, x, m)) = x = f_{[m'],i}(\sigma(i, x, m'))$. Let $[m] = m_1 \dots m_r$ and $[m'] = m'_1 \dots m'_{r'}$. Let l be the smallest index u with $m_u \neq m'_u$. Such an index l exists, since neither $[m]$ is a prefix of $[m']$ nor vice versa. We have $f_{m_1 \dots m_r, i}(\sigma(i, x, m)) = f_{m'_1 \dots m'_{r'}, i}(\sigma(i, x, m'))$, since $f_{0,i}$ and $f_{1,i}$ are injective. Then $f_{m_{l+1} \dots m_r, i}(\sigma(i, x, m))$ and $f_{m'_{l+1} \dots m'_{r'}, i}(\sigma(i, x, m'))$ are a claw of (f_0, f_1) .
- b. A successful existential forgery by a key-only attack computes a valid signature $\sigma(i, x, m)$ from (i, x) , for some message m . Let b be the first bit of $[m]$, i.e., $[m] = bm'$. Then $f_{m',i}(\sigma(i, x, m)) = f_{b,i}^{-1}(x)$. Thus, a pre-image of x may be computed from (i, x) , for $f_{0,i}$ or $f_{1,i}$. We obtain a contradiction to the one-way property of f_0 and f_1 .
- c. Adaptively-chosen-message attack means in a one-time signature scheme that the adversary knows the signature σ of one message m of his choice and tries to forge the signature σ' for another message m' .
Assume that a successful forger F exists performing an adaptively-chosen-message attack. Then, we can define an algorithm A which computes claws of f_0, f_1 with a non-negligible probability.
On input $i \in I_k$, A works as follows:
- i. Randomly choose a message $\tilde{m} \stackrel{u}{\leftarrow} \{0, 1\}^{c \lceil \log_2(k) \rceil}$.
 - ii. Randomly choose $x \stackrel{u}{\leftarrow} D_i$ and compute $z := f_{[\tilde{m}],i}(x)$.
 - iii. Call $F(i, z)$ with the key (i, z) .
Note that z is also uniformly distributed in D_i , since x was chosen uniformly and $f_{[\tilde{m}],i}$ is bijective.
 - iv. $F(i, z)$ requests the signature σ for a message m . If $m = \tilde{m}$ (which happens with probability $\geq 1/k^c$), then $\sigma = x$ is supplied to F . Otherwise, A returns with a failure.
 - v. If $F(i, z)$ now returns a valid forged signature σ' for a message $m' \neq m$, then A easily finds a claw of f_0, f_1 , as shown in a).
 A 's probability of success is greater than or equal to F 's probability of success multiplied by $1/k^c$, hence non-negligible. This is a contradiction.
- d. We do not know how to simulate the legitimate signer and provide the forger with the requested signature, with a non-negligible probability. The approach of c), simply to guess the message in advance, does not work, if there are exponentially many messages.
6. a) The verification procedure for a signature $\sigma = (s, \hat{m})$ for m is:
1. Check whether \hat{m} is well-formed, i.e., $\hat{m} = [\hat{m}_1] \parallel \dots \parallel [\hat{m}_r]$ with messages $m_j \in \{0, 1\}^*$.

2. Check $f_{\hat{m}\| [m], i}(s) = x$.

The first step cannot be omitted, in general. Take, e.g., the prefix-free encoding given in Section 10.2, and assume that an adversary learns a valid signed message $(m, (s, \hat{m}))$, $\hat{m} = [m_1]\| \dots \| [m_r]$. Let t be a proper tail of m (i.e., $m = \tilde{u}\| t$, $t \neq m$), and let u be defined by $[m] = u\| [t]$. Then $(s, \hat{m}\| u)$ is a valid signature for t , i.e., it passes step 2 of the verification procedure.

b) Assume there is a probabilistic polynomial algorithm $F(i, x, m_1, \sigma_1, \dots, m_l, \sigma_l)$ that tries to forge a signature for $\tilde{m} \neq m_1, \dots, m_l$, when supplied with i, x and all the signatures for messages m_1, \dots, m_l that were generated before by the user with key (i, x) ($l = l(k)$ a polynomial function). Recall that the messages of user (i, x) are generated by the probabilistic polynomial algorithm $M(i)$. Assume that F is successful. This means that there is a positive polynomial, such that

$$\begin{aligned} \text{prob}(F(i, x, m_1, \sigma(i, x, m_1), \dots, m_l, \sigma(i, x, m_l))) &= (\tilde{m}, \tilde{\sigma}), \\ \text{Verify}(\tilde{m}, \tilde{\sigma}) &= \text{accept} : i \stackrel{u}{\leftarrow} I_k, x \stackrel{u}{\leftarrow} D_i, (m_1, \dots, m_l) \leftarrow M(i) \\ &> \frac{1}{P(k)}, \end{aligned}$$

for k in an infinite subset $\mathcal{K} \subseteq \mathbb{N}$.

We now define an algorithm \tilde{A} which, with non-negligible probability, either finds a claw of f_0, f_1 or inverts f_0 resp. f_1 . This will be the desired contradiction. \tilde{A} works on input i, x as follows:

- a. $(m_1, \dots, m_l) := M(i)$. Let $m := [m_1]\| \dots \| [m_l]$.
- b. Let $z := f_{m, i}(x) = f_{[m_1], i}(f_{[m_2], i}(\dots f_{[m_l], i}(x) \dots))$.
- c. Generate the signatures

$$\begin{aligned} \sigma(i, z, m_1) &= (f_{[m_2], i}(\dots f_{[m_l], i}(x), \varepsilon)) \\ \sigma(i, z, m_2) &= (f_{[m_3], i}(\dots f_{[m_l], i}(x), [m_1])) \\ &\dots \\ \sigma(i, z, m_l) &= (x, [m_1]\| \dots \| [m_{l-1}]). \end{aligned}$$

Here, ε denotes the empty string.

- d. $(\tilde{m}, \tilde{\sigma}) := F(i, z, m_1, \sigma(i, z, m_1), \dots, m_l, \sigma(i, z, m_l))$. We have $\tilde{\sigma} = (s, \hat{m})$ and $\tilde{m} \neq m_j, 1 \leq j \leq l$. If $(\tilde{m}, \tilde{\sigma})$ does not pass the verification procedure, we return some random element and stop. Otherwise, if $(\tilde{m}, \tilde{\sigma})$ is a valid signature, i.e., $f_{\hat{m}\| [\tilde{m}], i}(s) = z$, we continue.
- e. $\hat{m}\| [\tilde{m}]$ is not a prefix of m , because otherwise \tilde{m} would be equal to one of the m_j . Here, note that by step 1 in the verification procedure, \hat{m} is well-formed with respect to the prefix-free encoding. The algorithm now distinguishes two cases.

f. Case 1: m is not a prefix of $\hat{m} \parallel [\tilde{m}]$.

We have $\sigma(i, z, m_l) = (x, \dots)$. Since $f_{m,i}(x) = z = f_{\hat{m} \parallel [\tilde{m}]}(s)$, we immediately find a claw of $f_{0,i}, f_{1,i}$, as in Exercise 6 a). We return this claw.

g. Case 2: m is a prefix of $\hat{m} \parallel [\tilde{m}]$.

Let $\hat{m} \parallel [\tilde{m}] = m \parallel u$. Note that $u \neq \varepsilon$, because $\tilde{m} \neq m_l$. Let $u = bu', b \in \{0, 1\}$. Since $f_{u,i}(s) = f_{m,i}^{-1}(z) = x$, we can immediately compute $f_{b,i}^{-1}(x) = f_{u',i}(s)$. We return this pre-image.

We have

$$\begin{aligned} \text{prob}(A(i, x) \text{ is a claw or one of the pre-images } f_{0,i}^{-1}(x), f_{1,i}^{-1}(x) : \\ & \qquad \qquad \qquad i \stackrel{u}{\leftarrow} I_k, x \stackrel{u}{\leftarrow} D_i) \\ &= \text{prob}(F(i, f_{m,i}(x), m_1, \sigma(i, f_{m,i}(x), m_1), \dots, m_l, \sigma(i, f_{m,i}(x), m_l)) \\ & \qquad \qquad \qquad = (\tilde{m}, \tilde{\sigma}), \\ & \qquad \qquad \text{Verify}(\tilde{m}, \tilde{\sigma}) = \text{accept} : \\ & \qquad \qquad \qquad i \stackrel{u}{\leftarrow} I_k, (m_1, \dots, m_l) \leftarrow M(i), x \stackrel{u}{\leftarrow} D_i) \\ &= \text{prob}(F(i, z, m_1, \sigma(i, z, m_1), \dots, m_l, \sigma(i, z, m_l)) = (\tilde{m}, \tilde{\sigma}), \\ & \qquad \qquad \text{Verify}(\tilde{m}, \tilde{\sigma}) = \text{accept} : i \stackrel{u}{\leftarrow} I_k, (m_1, \dots, m_l) \leftarrow M(i), z \stackrel{u}{\leftarrow} D_i) \\ &> \frac{1}{P(k)}, \end{aligned}$$

for infinitely many k , a contradiction to the claw-freeness and the one-way property of f_0, f_1 .

Note that $f_{m,i}$ is a permutation of D_i and hence $z = f_{m,i}(x), x \stackrel{u}{\leftarrow} D_i$ is the uniform distribution $z \stackrel{u}{\leftarrow} D_i$.