Problem sheet 1 for Course 02231, 2025

These practice problems have the purpose of helping you understand the material better and learning the skills that are necessary to analyze cryptographic constructions, and sometimes to prepare you for the next class. All answers should be supported by a written justification. To gauge whether a justification is sufficient, ask yourself if your peers would be convinced by it without additional explanations.

Exercise 1.1 This exercise is in preparation for the second lecture. We start by recapping some notation. A random variable X on a finite set $\mathcal{V}$ is an element of $\mathcal{V}$ chosen at random according a probability distribution $D : \mathcal{V} \to \mathbb{R}$, i.e. a function such that $D(x) \geq 0$ for all $x \in \mathcal{V}$ and

$$\sum_{x \in \mathcal{V}} D(x) = 1.$$

We write $X \leftarrow D$ to mean "X is sampled according to D". If D is the uniform distribution, i.e. $D(x) = \frac{1}{|\mathcal{V}|}$ for all x, then we also write $X \leftarrow \mathcal{V}$.

a) Let $X_1, X_2, X_3, X_4 \leftarrow \{1, 2, 3, 4, 5, 6\}$ be four throws of a fair die. Compute the probability that $X_i = 6$ for at least one $i \in \{1, 2, 3, 4\}$. Write down your calculation using (some of) the notation introduced above.

b) Your friend offers you the following bet: Before your friend tosses a coin 10 times (denote "heads" by 0 and "tails" by 1) you can either try to guess the first 6 coin tosses, or you can guess a sequence of 7 coin toss results where you think it will be a sub-sequence of the tosses (Example: if you guessed $0, 0, 1, 0, 1, 1, 1$ and the tosses come up $1, 0, \mathbf{0}, \mathbf{0}, \mathbf{1}, \mathbf{0}, \mathbf{1}, \mathbf{1}, \mathbf{1}, 0$ you win). If you win, you get DKK50, if you lose you have to pay DKK1. Describe a strategy and compute its winning probability. Is your strategy the best strategy? If so, why? Would you take the bet?

Exercise 1.2 Here is another problem about probability, to practice thinking about independence.

1. Here are three joint probability distributions p_i, $i = 1, 2, 3$, on $\{0,1\} \times \{0,1\}$ given as matrices, the upper left entry is $p_i(0,0)$, the upper right entry is $p_i(0,1)$ etc. Let $(X_i, Y_i) \leftarrow p_i$ be random variables with joint distribution p_i. Which of the pairs are independent?

$$p_1 : \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{pmatrix}$$
$$p_2 : \begin{pmatrix} \frac{1}{3} & \frac{1}{3} \\ 0 & \frac{1}{3} \end{pmatrix}$$
$$p_3 : \begin{pmatrix} \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} \end{pmatrix}$$

Exercise 1.3 We can generalize the Caesar cipher from the lecture in the following straightforward way: consider the alphabet $\mathcal{X} = \{x_0, \ldots, x_{\ell-1}\}$ over which plaintext and ciphertext

space are defined. The key is a uniformly random number $k \in \{0, \ldots, \ell - 1\}$. The encryption of a string $m = x_{i_0} x_{i_1} \ldots x_{i_{M-1}}$ of letters in $\mathcal{X}$ replaces each character x_i by x_j with $j = i + k \bmod \ell$, i.e.

$$e_k(m) = x_{i_0+k \bmod \ell} x_{i_1+k \bmod \ell} \ldots x_{i_{M-1}+k \bmod \ell}.$$

Decryption is given by $d_k(c) = e_{-k \bmod \ell}(c)$.

Let $\mathcal{Y} = \{y_0, \ldots, y_{127}\}$ be the ASCII character table `https://simple.m.wikipedia.org/wiki/File:ASCII-Table-wide.svg`, so we have for example $y_0 =$ `NULL`, $y_{65} =$ `A` and $y_{49} =$ `1`. For any $a, b \in \{0, \ldots, 127\}$, define $\mathcal{X}_{a,b} = \{x_0, \ldots, x_{b-a}\}$ with $x_i = y_{i+a}$. For any such alphabet $\mathcal{X}_{a,b}$ we can consider the Caesar cipher on it. For some $a, b \in \{0, \ldots, 127\}$, the following is a ciphertext that was generated by encrypting a message string m consisting of English text with characters from $\mathcal{X}_{a,b}$ only, using the Caesar cipher on $\mathcal{X}_{a,b}$:

```
;\r6TXfTe~r[bjrTeXrlbhrWb\aZ2rHf\aZrUeb^XarVelcgb~r[h[2r;TccXafrgbrg[X
rUXfgrbYrhf!!!rAXkgrg\‘XrTebhaW~rgelr48F $%+r\ar:T_b\fr6bhagXer@bWXsss
```

Find the plaintext m, the values a and b, and the key!

Hints:

1. You might need to "escape" some characters when handing the ciphertext string to a program. The ciphertext as given above does not contain any escape sequences.
2. You can solve this problem by "brute force". In that case, the challenge is to find a way to automatically check whether a string of ASCII characters is English text. Another option is to look at the most frequent characters ("frequency analysis").
3. The plaintext is regular English text. In particular, it has spaces.

Exercise 1.4 Practice applying Kerckhoffs' principle. To do that, pick a physical security system (door lock, camera surveillance, boarding passes...) and analyze in howfar common instances of the system fulfil Kerckhoffs' principle.

Exercise 1.5 The last exercise is more for your fun: to read up a bit on the history of cryptography and how it sometimes fails.

Here are the three examples from the introductory lecture:

1. The Battle of Midway: `https://www.nsa.gov/portals/75/documents/about/cryptologic-heritage/historical-figures-publications/publications/wwii/battle-midway.pdf`
2. The history of the German ENIGMA: `https://www.dpma.de/english/our_office/publications/milestones/computerpioneers/enigma/index.html`
3. CCA security is not just a joke: `https://blog.cryptographyengineering.com/2016/03/21/attack-of-week-apple-imessage/`

There are of course many interesting websites and books about cryptography and its history. A somewhat outdated, but still interesting book is *The Codebreakers* by David Kahn.

Problem sheet 2 for Course 012231, 2025

These practice problems have the purpose of helping you understand the material better and learning the skills that are necessary to analyze cryptographic constructions, and sometimes to prepare you for the next class. All answers should be supported by a written justification. To gauge whether a justification is sufficient, ask yourself if your peers would be convinced by it without additional explanations.

We denote vectors as $\mathbf{x} \in \{0,1\}^\lambda$. By $\mathbf{x}[i]$ we denote the ith index of $\mathbf{x}$, where $i \in \{1,\ldots,\lambda\}$. As in the lecture, we write $k \leftarrow K$ if k is sampled from the set K such that it can be each element from K with equal probability $1/|K|$.

For the first exercise, we denote with & the AND-function, which is defined by the following truth table:

x	y	$x\&y$
0	0	0
0	1	0
1	0	0
1	1	1

For two vectors $\mathbf{x}, \mathbf{y}$ we can define the coordinate-wise AND of both vectors, $\mathbf{x}\&\mathbf{y}$, by saying that $(\mathbf{x}\&\mathbf{y})[i] = \mathbf{x}[i]\&\mathbf{y}[i]$. This means that $\mathbf{x}\&\mathbf{y}$ is computed by applying & to the individual coordinates of each $\mathbf{x}, \mathbf{y}$ and concatenating the outcomes.

Exercise 1. (**Not all bit operations are equal**)

Let λ be a positive integer. We define $K = M = C = \{0,1\}^\lambda$ and consider the following algorithms:

KEYGEN()**:**

1. Output $\mathbf{k} \leftarrow K$.

ENC($\mathbf{k}, \mathbf{m} \in M$)**:**

1. Output $\mathbf{k}\&\mathbf{m}$.

DEC($\mathbf{k}, \mathbf{c} \in C$)**:**

1. Output $\mathbf{k}\&\mathbf{c}$.

Show that these three algorithms do *not* form a SKE scheme, because they are not correct. You can show this by giving an example where decryption fails.

For the second exercise, we will work with permutations on bitstrings. A permutation of a string is a rearrangement of all the entries of that string. For example, let $a, b \in \{0,1\}$ be bits and ab be a bitstring. Then both ab and ba are permutations of the original string. Note that aa is not a permutation of ab, as b also has to appear in the permutation of ab. In general, there are $n!$ permutations of n objects. For an n-bit string, there are thus $n!$ permutations (but they are not all distinct).

Mapping all bitstrings ab to ab is a function from $\{0,1\}^2$ to $\{0,1\}^2$, and similarly is mapping all bitstrings ab to ba. For all strings of length n, we denote the set of permutations (or rather, all different functions that shuffle the n entries of input vectors) as S_n.

For a string $\mathbf{x} \in \{0,1\}^n$ we write $\mathbf{y} := \pi(\mathbf{x})$ to say that $\mathbf{y}$ is the string $\mathbf{x}$ permuted under the permutation function $\pi \in S_n$. Formally, this means that $\mathbf{y}[\pi(i)] = \mathbf{x}[i]$ for all $i \in \{1, \ldots, n\}$. Accordingly, let π^{-1} be the inverse permutation. Note that this inverse permutation always exists: if we shuffle the indices of a vector in a certain way, we can just unshuffle them by going backwards.

Exercise 2. (**Permutations for Security?**)

Let λ be a positive integer and let S_λ be the set of all permutations on strings of length λ. We define $M = C = \{0,1\}^\lambda$ and consider the following algorithms:

KEYGEN()**:**

1. Output $\pi \leftarrow S_\lambda$.

ENC$(\pi, \mathbf{m} \in M)$**:**

1. Output $\pi(\mathbf{m})$.

DEC$(\pi, \mathbf{c} \in C)$**:**

1. Output $\pi^{-1}(\mathbf{c})$.

1. Show that these three algorithms form an SKE scheme, i.e. that they are correct.
2. Let L_{OTS-0}, L_{OTS-1} be the two worlds in the left-or-right one-time security game described in the lecture and the book. Show that the defined three algorithms are not one-time indistinguishable, i.e. that $L_{OTS-0} \not\equiv L_{OTS-1}$. For this, write down an explicit algorithm B which sends one query to the library that it has access to black box. Based on the response c, B outputs 0 if it thinks that it talked to L_{OTS-0} or 1 if it thinks that it talked to L_{OTS-1}. How likely is it that B gives the right answer?

For the last exercise, we denote with $\oplus$ the XOR-function, which is defined by the following truth table:

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

As for the AND-function, we can define it coordinate-wise for vectors.

Exercise 3. (**Why** KEYGEN **is important**)

Let λ be a positive integer. We define $K = M = C = \{0,1\}^\lambda$ and let $\mathcal{B}_p$ be the probability distribution that outputs 0 with probability p and 1 with probability $1-p$. Consider the following algorithms:

KEYGEN()**:**

1. Let $\mathbf{k}$ be a vector of length λ.
2. For each $i \in \{1, \ldots, \lambda\}$ set $\mathbf{k}[i] \leftarrow \mathcal{B}_{0.75}$

ENC($\mathbf{k}, \mathbf{m} \in M$)**:**

1. Output $\mathbf{k} \oplus \mathbf{m}$.

DEC($\mathbf{k}, \mathbf{c} \in C$)**:**

1. Output $\mathbf{k} \oplus \mathbf{c}$.

1. Show that these three algorithms form an SKE scheme, i.e. that they are correct.
2. Let $\lambda = 3$ and let A be an algorithm which queries the $L_{OTS-Real}$ library with input $\mathbf{m} = 111$. Compute the probability of each possible ciphertext obtained from the library.
3. Based on the algorithm A, consider a new algorithm B that sends $\mathbf{m} = 111$, waits for the response $\mathbf{c}$ from the library. B outputs 1 if $\mathbf{c} = \mathbf{m}$ and 0 otherwise. What is the probability that B outputs 1 with $L_{OTS-Real}$, and what is the probability when it interacts with $L_{OTS-Rand}$?
4. Show that the three algorithms KEYGEN, ENC, DEC are not real-or-random secure, i.e. that $L_{OTS-Real} \not\equiv L_{OTS-Rand}$. For this, apply the real-or-random-secure definition to the algorithm B and show why it breaks security.
5. * **Bonus problem:** Is B the attacker with the highest probability of guessing correctly? If not, find an optimal attack.

Problem sheet 3 for Course 02231, 2025

These practice problems have the purpose of helping you understand the material better and learning the skills that are necessary to analyze cryptographic constructions, and sometimes to prepare you for the next class. All answers should be supported by a written justification. To gauge whether a justification is sufficient, ask yourself if your peers would be convinced by it without additional explanations.

We denote vectors as $\mathbf{x} \in \{0,1\}^\lambda$. By $\mathbf{x}[i]$ we denote the ith index of $\mathbf{x}$, where $i \in \{1,\ldots,\lambda\}$. As in the lecture, we write $k \leftarrow K$ if k is sampled from the set K such that it can be each element from K with equal probability $1/|K|$.

Exercise 1. (**Triple the DES!**)

In the lecture you learned about the DES Pseudorandom Permutation (PRP) or block cipher, which has block size $B = 64$ and key length $\lambda = 56$. We denote its algorithms as $\mathsf{DES.KEYGEN}, \mathsf{DES.ENC}, \mathsf{DES.DEC}$. A well-known variation of the DES algorithm is called Triple-DES and works as follows:

$\mathsf{3DES.KEYGEN}()$**:**

1. Compute $k_1, k_2, k_3 \leftarrow \mathsf{DES.KEYGEN}()$.
2. Output $k = (k_1, k_2, k_3)$.

$\mathsf{3DES.ENC}(k, m)$**:**

1. Check that $k = (k_1, k_2, k_3)$.
2. Output $\mathsf{DES.ENC}(k_3, \mathsf{DES.DEC}(k_2, \mathsf{DES.ENC}(k_1, m)))$.

1. Find the missing $\mathsf{3DES.DEC}$ algorithm and argue why the overall scheme is correct.
2. In a so-called brute-force attack, an attacker tries out all possible keys from the keyspace until it finds the one which decrypts a ciphertext. Assume that we are given a DES plaintext/ciphertext pair $m, c = \mathsf{DES.ENC}(k, m)$ for an unknown key k. Our goal is to find k. Further, assume for simplicity that one $\mathsf{DES.ENC}$ and thus $\mathsf{DES.DEC}$-operation takes 1ns of time on a computer of your choice.
   (a) How many days does it take to recover the one (or more) keys which would have led to the given plaintext/ciphertext pair using the computer?
   (b) Assume you have 1024 such machines at your disposal. Can you use these machines to speed up the brute-force attack? How long will the attack take now?
   (c) Compute the same attack runtime for 3DES. You may want to compare the runtime of the attack to the age of the universe, which is estimated at 13.8 billion years.

Exercise 2. (There are more PRFs)

In the class, you learned what a Pseudorandom Function, or PRF for short, is: It is a function $F : \{0,1\}^{\lambda} \times \{0,1\}^{in} \to \{0,1\}^{out}$ such that $L^F_{PRF-Real}$ and $L^F_{PRF-Rand}$ are indistinguishable.

Assume that you have such a PRF F given. Let $\mathbf{r} \in \{0,1\}^{out}$ be any fixed, publicly known string, and define the function $G : \{0,1\}^{\lambda} \times \{0,1\}^{in} \to \{0,1\}^{out}$ as $G(k,x) = F(k,x) \oplus \mathbf{r}$.

Show that G is also a PRF. To prove this, it is instructive to follow the steps below:

1. Write down the *Lookup* functions in the libraries $L^G_{PRF-Real}$ and $L^G_{PRF-Rand}$.
2. Find a way to rewrite $L^G_{PRF-Real}$ into another library $\overline{L}^G_{PRF-Real}$ which uses $L^F_{PRF-Real}$.
3. You can now apply the hybrid technique to switch out $L^F_{PRF-Real}$ with $L^F_{PRF-Rand}$.
4. Based on this, find an argument how to complete indistinguishability of the resulting library with $L^G_{PRF-Rand}$.

Exercise 3. (Birthday bounds in practice)

In the class, you learned about $BirthdayProb(q,N)$ as the probability of sampling two or more identical items after q random queries from a set of size N. In class, we computed an upper-bound on this probability.

1. Write a program (in your preferred programming language) that computes $BirthdayProb(q,N)$ given the exact expression that we used in the class. The program should also compute the upper-bound on $BirthdayProb(q,N)$ that we derived in class. How close is the upper-bound to the real value?
2. Compute the probability that two of your fellow students in this course (yourself included) are born on the same date, assuming uniform distribution of birthdays throughout the year, using your program.
3. In the class, we used $BirthdayProb(q,N)$ to argue that we sometimes can treat PRFs and PRPs as the same! But this assumed that $q \ll N$. Assume you have a PRF $F : \{0,1\}^{32} \times \{0,1\}^{32} \to \{0,1\}^{32}$ as well as a PRP $G : \{0,1\}^{32} \times \{0,1\}^{32} \to \{0,1\}^{32}$. What is the probability of distinguishing F from G for $q = 2, 2^8, 2^{12}$ or 2^{16}?

Problem sheet 4 for Course 02231, 2025

These practice problems have the purpose of helping you understand the material better and learning the skills that are necessary to analyze cryptographic constructions, and sometimes to prepare you for the next class. All answers should be supported by a written justification. To gauge whether a justification is sufficient, ask yourself if your peers would be convinced by it without additional explanations.

We denote vectors as $\mathbf{x} \in \{0,1\}^\lambda$. By $\mathbf{x}[i]$ we denote the ith index of $\mathbf{x}$, where $i \in \{1,\ldots,\lambda\}$. As in the lecture, we write $k \leftarrow K$ if k is sampled from the set K such that it can be each element from K with equal probability $1/|K|$.

Exercise 1. (Never use ECB)

In the lecture, it was mentioned that the ECB mode of operation should never be used for anything, under any circumstances. In this exercise we will see that it is not even CPA secure.

1. Assume ECB is used with a block cipher of block length B. Consider the encryptions of the messages $\underbrace{0\ldots0}_{B \text{ bits}}|\underbrace{1\ldots1}_{B \text{ bits}}$ and $\underbrace{0\ldots0}_{B \text{ bits}}|\underbrace{0\ldots0}_{B \text{ bits}}$. Can you see a problem?
2. Turn the discovered problem into an attacker against left-right CPA security. Your attacker should make 1 query to the library, and always succeed.

Exercise 2. (Padding for modes of operation)

In the lecture, we saw that modes of operation allow to encrypt messages of length $\ell \cdot B$ bits, where B is the block length and ℓ is an integer. Unfortunately, not every message is an exact multiple of B. To support messages of arbitrary lengh, a so-called *Padding scheme* is used.

The ANSI X.923 padding works as follows:

1. Let m be the message of n bytes length, such that $\ell = \lfloor 8n/B \rfloor$.
2. Output the message $m' = \underbrace{m|00|00|\ldots|00|b}_{(\ell+1)\cdot(B/8) \text{ bytes}}$, where b is the number of bytes that had to be added to m. b itself is encoded in one byte, and therefore has length 8 bits.

For example, if m is 4 bytes short of being a multiple of B, then one would append 00|00|00|04 as padding. If it is just missing two bytes, then we instead add 00|02.

1. Assume that m already has a length that is a multiple of B and that it ends with 01. Why do we have to add a whole block of length B to m before encrypting it, instead of just encrypting the original message?
2. What is the maximal block size (in bits) that can be supported by the ANSI X.923 padding scheme?
3. Consider the CTR mode of operation. It can be modified to be able to encrypt messages whose lengths are not multiples of B. How can this be done?

Exercise 3. (Adversarial Sampling)

In the class, it was claimed that any polynomial-time (in λ) attacker only has negligible (in λ) distinguishing advantage for the following libraries:

Library $L_{\textbf{GenSample}}$:

$\underline{Sample(R \subset \{0,1\}^\lambda)}$:

1. $r \leftarrow \{0,1\}^\lambda$
2. Output r

Library $L_{\textbf{GenSampleIgnore}}$:

$\underline{Sample(R \subset \{0,1\}^\lambda)}$:

1. $r \leftarrow \{0,1\}^\lambda \setminus R$
2. Output r

We will now show that this is true, *if the attacker has to write down every element of R before giving it to Sample.*

1. Devise a strategy how you would distinguish both libraries. Write down an adversary, as an algorithm, to do this. You can use one or more queries to *Sample*. Let's call this algorithm $\mathcal{A}$.

2. Assume your algorithm $\mathcal{A}$ makes 1 call to *Sample*, where $|R| = n$. For which outputs of $L_{\text{GenSample}}$ will $\mathcal{A}$ output 1 with the same probability as when $\mathcal{A}$ talks to $L_{\text{GenSampleIgnore}}$? And for which outputs of $L_{\text{GenSample}}$ will $\mathcal{A}$ output something different? How does this translate into the advantage of $\mathcal{A}$ making one query?

3. Assume that $\mathcal{A}$ makes 2 calls to *Sample*. What is maximal distinguishing advantage for $\mathcal{A}$ now?

4. Generalize your argument to $q \in \mathbb{N}$ calls to *Sample*. You can use the trick from the Birthday Bound in the lecture to give an upper bound on the success probability.

5. Assume that your attacker $\mathcal{A}$ makes $q \in \mathbb{N}$ calls to *Sample*, and that generating each item of R for each call takes 1 unit of time. Argue that since $\mathcal{A}$ has a runtime that is polynomial in λ, then this translates into a polynomial number of calls q and a polynomial size of R in each call (using the restriction on R outlined above). Then, conclude why the distinguishing advantage must be negligible.

Problem sheet 5 for Course 02231, 2025

These practice problems have the purpose of helping you understand the material better and learning the skills that are necessary to analyze cryptographic constructions, and sometimes to prepare you for the next class. All answers should be supported by a written justification. To gauge whether a justification is sufficient, ask yourself if your peers would be convinced by it without additional explanations.

As in the lecture, we write $k \leftarrow K$ if k is sampled from the set K such that it can be each element from K with equal probability $1/|K|$. For two strings x, y we use $x|y$ to denote the string obtained from concatenating x with y.

Exercise 1. (**Mix and Match for ECB-MAC**)

Assume that $\Theta = (Keygen, MAC)$ is a secure MAC scheme where $MAC : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$.
In the class, we mentioned why ECB-MAC is a terrible idea: one can simply swap blocks in the message (and the MAC) and the result is again a valid message! A quick fix is to encode the block number as part of the message. For example, consider the following algorithm where $m_1, m_2 \in \{0,1\}^{\lambda-1}$:

$Keygen()$**:**

1. Output $k \leftarrow \Theta.Keygen()$

$ECB - MAC'(k, m_1|m_2)$**:**

1. $t_1 \leftarrow \Theta.MAC(k, 0|m_1)$
2. $t_2 \leftarrow \Theta.MAC(k, 1|m_2)$
3. Output (t_1, t_2)

1. Assume that you obtain two outputs $(t_1, t_2) \leftarrow ECB - MAC'(k, 0^{2\lambda-2})$ and $(t'_1, t'_2) \leftarrow ECB - MAC'(k, 1^{2\lambda-2})$. Can you find a message m such that $(t_1, t'_2) == ECB - MAC'(k, m)$?
2. Using your previous observation, construct an attacker which can distinguish $L_{MAC-real}$ from $L_{MAC-fake}$ for $ECB - MAC'$ with probability 1. Your attacker should make two queries to *Gettag* and one to *Checktag*.

Exercise 2.

As in the previous exercise, assume that $\Theta = (Keygen, MAC)$ is a secure MAC scheme where $MAC : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$.
In the lecture, it was mentioned that CBC-MAC is a secure MAC for message space $M = \{0,1\}^{\lambda\ell}$. It is only secure *if all messages have the same length*. We will now see why this is the case. First, let us remember how $CBC - MAC$ works:

$Keygen()$**:**

1. Output $k \leftarrow \Theta.Keygen()$

$CBC - MAC(k, m_1|\ldots|m_\ell)$**:**

1. $t \leftarrow 0^\lambda$
2. For $i = 1, \ldots, \ell$
   (a) $t \leftarrow \Theta.MAC(k, t \oplus m_i)$
3. Output t

1. Assume that you set $\ell = 2$ and generate a MAC t for input $m_1|m_2$. What is the MAC of the message $m_1|m_2|(m_1 \oplus t)|m_2$?
2. Using the previous observation, construct an attacker which can distinguish $L_{MAC-real}$ from $L_{MAC-fake}$ for CBC-MAC with probability 1. Your attacker should make one query to *Gettag* and one to *Checktag*.

Exercise 3.

In this exercise, we will prove that *Encrypt-then-MAC* is a CCA-secure encryption scheme. For this, we assume that $\Omega = (Keygen, Enc, Dec)$ is a CPA-secure encryption scheme (for the left-right definition), that $\Theta = (Keygen, MAC)$ is a secure MAC scheme and that $\Omega.C \subseteq \Theta.M$ (meaning that the ciphertexts generated by Ω are valid messages which can be MACed by Θ). To recap, the resulting MAC-then-encrypt scheme Σ looks as follows:

$\Sigma.Keygen()$**:**

1. $k_E \leftarrow \Omega.Keygen()$
2. $k_M \leftarrow \Theta.Keygen()$
3. Output (k_E, k_M)

$\Sigma.Enc((k_E, k_M), m \in \Omega.M)$**:**

1. $c \leftarrow \Omega.Enc(k_E, m)$
2. $t \leftarrow \Theta.MAC(k_M, c)$
3. Output (c, t)

$\Sigma.Dec((k_E, k_M), (c, t))$**:**

1. If $t \neq \Theta.MAC(k_M, c)$ then output $\perp$
2. Output $\Omega.Dec(k_E, c)$

Our goal in the proof is to show that the libraries L^{Σ}_{CCA-0} and L^{Σ}_{CCA-1} are indistinguishable for any polynomial-time attacker. Here, we define L^{Σ}_{CCA-i} as

L^{Σ}_{CCA-i}**:**

$k \leftarrow \Sigma.Keygen()$

$S \leftarrow \varnothing$

$CTXT(m_0, m_1 \in \Sigma.M)$**:**

1. If $|m_0| \neq |m_1|$ then output $\perp$
2. $c \leftarrow \Sigma.Enc(k, m_i)$
3. $S \leftarrow S \cup \{c\}$
4. Ouptut c

$Decrypt(c \in \Sigma.C)$**:**

1. If $c \in S$ then output $\perp$
2. Output $\Sigma.Dec(k, c)$

1. Check that the encryption scheme Σ is correct. For this, you can assume that Ω is correct.

2. Write down the library L^{Σ}_{CCA-0} where you replace $\Sigma.Keygen, \Sigma.Enc, \Sigma.Dec$ with the algorithms that implement Σ, i.e. the algorithms from Θ and Ω. For example, you replace the line $k \leftarrow \Sigma.Keygen()$ with the two lines $k_E \leftarrow \Omega.Keygen()$ and $k_M \leftarrow \Theta.Keygen()$...

3. Write down the library L_{Step-1}. L_{Step-1} should be the same as L^{Σ}_{CCA-0}, but use a library $L^{\Theta}_{MAC-real}$ to perform the MAC computations instead of having the MAC algorithms being run inside L_{Step-1}. Argue why $L^{\Sigma}_{CCA-0} \equiv L_{Step-1} \circ L^{\Theta}_{MAC-real}$.

4. Argue why $L_{Step-1} \circ L^{\Theta}_{MAC-real} \approx L_{Step-1} \circ L^{\Theta}_{MAC-fake}$.

5. You can now rewrite $L_{Step-1} \circ L^{\Theta}_{MAC-fake}$ into a new library L_{Step-2} which does not use $L^{\Theta}_{MAC-fake}$ anymore (so the code of $L^{\Theta}_{MAC-fake}$ now resides inside L_{Step-2}). Argue why $L_{Step-1} \circ L^{\Theta}_{MAC-fake} \equiv L_{Step-2}$.

6. In L_{Step-2} you still have $\Omega.Dec(k_E, c)$ inside $Decrypt$. Consider the library L_{Step-3} which is the same as L_{Step-2}, but without computing $\Omega.Dec(k_E, c)$. Why is $L_{Step-2} \equiv L_{Step-3}$? For this, you can check under which conditions $\Omega.Dec(k_E, c)$ will be reached.

7. You can now construct a library L_{Step-4} which itself uses L^{Ω}_{CPA-0} to do the encryption. Argue that $L_{Step-3} \equiv L_{Step-4} \circ L^{\Omega}_{CPA-0}$.

8. Finally, show that $L_{Step-4} \circ L^{\Omega}_{CPA-0} \approx L_{Step-4} \circ L^{\Omega}_{CPA-1}$. Describe how to finish the proof from here.

Problem sheet 6 for Course 02231, 2025

These practice problems have the purpose of helping you understand the material better and learning the skills that are necessary to analyze cryptographic constructions, and sometimes to prepare you for the next class. All answers should be supported by a written justification. To gauge whether a justification is sufficient, ask yourself if your peers would be convinced by it without additional explanations.

We denote vectors as $\mathbf{x} \in \{0,1\}^\lambda$. By $\mathbf{x}[i]$ we denote the ith index of $\mathbf{x}$, where $i \in \{1,\ldots,\lambda\}$. As in the lecture, we write $k \leftarrow K$ if k is sampled from the set K such that it can be each element from K with equal probability $1/|K|$. For two strings x, y we use $x|y$ to denote the string obtained from concatenating x with y.

Exercise 1. (**Length extension attacks for Merkle-Damgård**)

Let $h : \{0,1\}^{\ell+\lambda} \to \{0,1\}^\lambda$ be a collision-resistant compression function.
In the class, we mentioned the Merkle-Damgård construction for hashes. Let us quickly recap how it works:

$MDPad_\ell(m)$**:**

1. Let e be $|m|$ in binary form, where e consists of ℓ bits.
2. While $|m|$ is not a multiple of ℓ: $m := m|0$.
3. Output $m|e$

$MDHash(m)$**:**

1. Let $m_1,\ldots,m_\tau = MDPad_\ell(m)$ where m_i has length ℓ bits.
2. Set $y_0 := 0^\lambda$.
3. For $i = 1,\ldots,\tau$: $y_i = h(m_i|y_{i-1})$.
4. Output y_τ.

1. Consider that we want to use the Merkle-Damgård construction as a MAC. For this, we redefine $MDHash(m)$ (which samples $y_0 \in \{0,1\}^\lambda$) as $MDMAC(k,m)$ which works like $MDHash(m)$, but instead sets $y_0 := k$. Show that given $(m, MDMac(k,m))$ it is easy to find (m', t') such that $t' = MDMac(k,m')$ where $m \neq m'$.

   Hint: The easiest way is to derive m' from m by making it a little bit longer. Maybe the function $MDPad_\ell$ can help you...

❓ Exercise 2. (Simplify Merkle-Damgård?)

Let $h : \{0,1\}^{\ell+\lambda} \to \{0,1\}^{\lambda}$ be a collision-resistant compression function.
Let us define a simplified version of Merkle-Damgård as follows:

$SMDPad_\ell(m)$**:**

1. If $|m| \leq \ell + \lambda$ then output $m|0^{\ell+\lambda-|m|}$.
2. Else Let $m = m_1|m_2$ where m_1 is of length λ.
3. While $|m_2|$ is not a multiple of ℓ: $m_2 := m_2|0$.
4. Output $m_1|m_2$

$SMDHash(m)$**:**

1. Let $m_1, \ldots, m_\tau = SMDPad_\ell(m)$ where m_1 has length λ bits and $m_2, \ldots, m_\tau$ ℓ bits
2. $y_1 = m_1$
3. For $i = 2, \ldots, \tau$: $y_i = h(m_i|y_{i-1})$.
4. Output y_τ.

1. Show that this simplified construction of Merkle-Damgård is not collision-resistant and construct two inputs m, m' such that $m \neq m'$ while $SMDHash(m) = SMDHash(m')$.

   Hint: m and m' do not have to be of the same length. If m is the longer message, then a starting point might be to choose a part of m' based on y_2 that is generated during $SMDHash(m)$...

❓ Exercise 3. (No homomorphism in Collision-resistant hashing)

Let $H : \{0,1\}^* \to \{0,1\}^{\lambda}$ be a function such that for all strings x, y of identical length, it holds that $H(x) \oplus H(y) = H(x \oplus y)$. This means that computing the hash of the coordinate-wise XOR of x and y is the same as hashing x and y separately using H and XOR-ing the resulting strings. Cryptographers would refer to such a function H as being *homomorphic*.

1. Show that H cannot be collision-resistant, by constructing an adversary that can distinguish $L^H_{CR-real}$ from $L^H_{CR-fake}$ (as defined in the lecture) with probability essentially 1.

   Hint: Consider the values $H(1) \oplus H(1)$ or $H(11) \oplus H(11)$...

Exercise 4. (**PRPs and collision resistance**)

Let $F : \{0,1\}^\lambda \times \{0,1\}^B \to \{0,1\}^B$ be a pseudorandom permutation on blocks of length B. Define the function $H : \{0,1\}^{\lambda+B} \to \{0,1\}^B$ as $H(x|y) := F(x,y)$ where x is of length λ and y of length B. This means that x is the key of the PRP F while y is the input to the permutation.

1. Show that H cannot be collision-resistant, by constructing an adversary that can distinguish $L^H_{CR-real}$ from $L^H_{CR-fake}$ (as defined in the lecture) with probability essentially 1.

   Hint: F is a pseudorandom permutation. Which operations are efficiently computable according to the definition?

Problem sheet 7 for Course 02231, 2025

These practice problems have the purpose of helping you understand the material better and learning the skills that are necessary to analyze cryptographic constructions, and sometimes to prepare you for the next class. All answers should be supported by a written justification. To gauge whether a justification is sufficient, ask yourself if your peers would be convinced by it without additional explanations.

We denote vectors as $\mathbf{x} \in \{0,1\}^\lambda$. By $\mathbf{x}[i]$ we denote the ith index of $\mathbf{x}$, where $i \in \{1,\ldots,\lambda\}$. As in the lecture, we write $k \leftarrow K$ if k is sampled from the set K such that it can be each element from K with equal probability $1/|K|$. For two strings x, y we use $x|y$ to denote the string obtained from concatenating x with y.

Exercise 1. (**Implementing RSA**)

We recap the RSA algorithm from the lecture:

$Keygen()$**:**

1. Sample two different odd prime numbers p, q and set $N = p \cdot q$.
2. Set $\varphi(N) = (p-1)\cdot(q-1)$ and choose e such that $gcd(e, \varphi(N)) = 1$. Then compute $d = e^{-1} \bmod \varphi(N)$.
3. Output $pk = (e, N)$ as public key and $sk = (d, N)$ as private key.

$Enc(pk, m)$**:** To encrypt the message $m \in \mathbb{Z}_N^*$ using public key $pk = (e, N)$:

1. Compute $c = m^e \bmod N$
2. Output c.

$Dec(sk, c)$**:** To decrypt a ciphertext $c \in \mathbb{Z}_N^*$ using private key $sk = (d, N)$:

1. Compute $m = c^d \bmod N$
2. Output m.

1. Implement the three algorithms constituting RSA in a programming language of your choice. For simplicity, you may hard-code the primes for key generation into the algorithm (i.e. pick them in advance). Make sure that p, q are sufficiently large, i.e. that their size is $\geq 2^{100}$ each.
2. Test that your implementation yields a correct cryptosystem. Moreover, add code to measure how fast encryption and decryption are.

Exercise 2. (The (Simple) Chinese Remainder Theorem)

Assume that we have 2 equations

- $x = a_1 \bmod b_1$
- $x = a_2 \bmod b_2$

Then the Chinese Remainder Theorem (CRT) states that there exists a unique solution for $x \bmod b_1 \cdot b_2$ if and only if $gcd(b_1, b_2) = 1$.

For example: Let $x = 3 \bmod 7$ and $x = 5 \bmod 9$, then since $gcd(7, 9) = 1$ they must have a unique solution.

To find the solution, compute $T \leftarrow b_1^{-1} \bmod b_2$, let $u \leftarrow (a_2 - a_1) \cdot T \bmod b_2$ and set $x \leftarrow a_1 + u \cdot b_1$.

1. For the example $x = 3 \bmod 7$ and $x = 5 \bmod 9$ find the unique solution $x \bmod 63$.
2. Show that this algorithm always terminates if $gcd(b_1, b_2) = 1$ and that its output is correct.

Exercise 3. (The Chinese Remainder Theorem and RSA)

When using RSA in practice, one usually chooses $e = 2^{16} + 1$ independent of N so e is small and prime. Since it is small, encrypting will only need 17 multiplications modulo N. Moreover, since e is prime it will almost always be coprime to $\varphi(N)$.

1. Show that you can compute $c = m^e \bmod N$ with only 17 multiplications modulo N.
2. Unfortunately, this means that d may now be a very large number, so decryption will take a long time. For example, if p, q are each 1000 bits long then N is around 2000 bits long so we may have to perform many operations modulo an integer of size 2000 bits. Show that, assuming $Keygen$ outputs $sk' = (d, p, q)$ instead of (d, N), then you can do a different, possibly faster decryption using the Chinese Remainder Theorem!
3. Update your encryption and decryption algorithm from Exercise 1 to use the fixed exponent $e = 2^{16} + 1$ and the new decryption method. Measure how fast your new encryption and decryption code is and compare to your implementation of Exercise 1.
4. **Bonus question:** If N has 2000 bits then also $\varphi(N)$ will have the same length (or maybe 1 bit shorter). Assuming e=$2^{16} + 1$, show that in this case d requires > 1980 bits to write it down, i.e. $d > 2^{1980}$.

Exercise 4. (**Finding $\varphi(N)$ implies factoring**)

Assume you have an algorithm A which, on input N, outputs the value $\varphi(N)$. Then show that any such algorithm A can efficiently factor RSA numbers in approximately the same runtime.

Your approach may look as follows:

1. Show that knowledge of $\varphi(N)$ and N allows you to reconstruct $p + q$. To see this, look at how $\varphi(N)$ is defined if $N = p \cdot q$ for two different primes p, q
2. Consider the polynomial $f(X) = (X - p) \cdot (X - q)$. What do you know about it's coefficients and how can you compute its roots?

Problem sheet 8 for Course 02231, 2025

These practice problems have the purpose of helping you understand the material better and learning the skills that are necessary to analyze cryptographic constructions, and sometimes to prepare you for the next class. All answers should be supported by a written justification. To gauge whether a justification is sufficient, ask yourself if your peers would be convinced by it without additional explanations.

We denote vectors as $\vec{x} \in \{0,1\}^\lambda$. By $\vec{x}[i]$ we denote the ith index of $\vec{x}$, where $i \in \{1,\ldots,\lambda\}$. As in the lecture, we write $k \leftarrow K$ if k is sampled from the set K such that it can be each element from K with equal probability $1/|K|$. For two strings x, y we use $x|y$ to denote the string obtained from concatenating x with y. Throughout this problem sheet, let p be a prime.

Recall the definition of the discrete Gaussian probability distribution

$$q_\sigma(z) = \frac{d_\sigma(z)}{\nu_\sigma},$$

where

$$d_\sigma(z) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{z^2}{2\sigma^2}}$$

is the probability density of the normal distribution $\mathcal{N}(0,\sigma^2)$ with mean 0 and variance σ^1, and

$$\nu_\sigma = \sum_{z\in\mathbb{Z}} d_\sigma(z).$$

Some helpful facts about the discrete Gaussian distribution:

1. $\nu_\sigma \geq 1 - d_\sigma(0)$
2. For a random variable $X \sim q_\sigma$ the following tail bound holds. For any integer $x > 1$,

$$\Pr[X \geq x] \leq \frac{\sigma}{\nu \cdot (x-1) \cdot \sqrt{2\pi}} e^{-\frac{(x-1)^2}{2\sigma^2}}.$$

For convenience, we quickly recap the Regev public-key encryption scheme:

Keygen()**:**

1. Sample $\mathbf{A} \leftarrow \mathbb{Z}_p^{\ell\times n}$ and $\mathbf{s} \leftarrow \mathbb{Z}_p^n$ uniformly at random.
2. Let $\mathbf{e} \in \mathbb{Z}_p^\ell$ where each e_i is distributed according to q_σ.
3. Set $\mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e} \bmod p$.
4. Output $pk = (\mathbf{A}, \mathbf{b})$ as public key and $sk = (\mathbf{s})$ as private key.

Enc(pk, m)**:** To encrypt the message $m \in \{0,1\}$ using public key $pk = (\mathbf{A}, \mathbf{b})$:

1. Let $\mathbf{r} \in \{0,1\}^\ell$ be uniformly random.
2. Set $\mathbf{c}_0 \leftarrow \mathbf{r}^\top \mathbf{A} \bmod p$ and $c_1 = \mathbf{r}^\top \mathbf{b} + m \cdot \frac{p-1}{2}$

3. Output $c = (\mathbf{c}_0, c_1)$.

Dec(sk, c)**:** To decrypt a ciphertext $c = (\mathbf{c}_0, c_1)$ using private key $sk = (\mathbf{s})$:

1. Compute $m' = \lfloor \frac{2(c_1 - \mathbf{c}_0^\top \mathbf{s}) \bmod p}{p} \rceil$
2. Output m'.

Exercise 1. (**Decryption failure probability**)

In this exercise, we derive a bound on the probability that the encryption algorithm of Regev's encryption scheme, on input a message m produces a ciphertext c such that the decryption algorithm decrypts c to a different message m'. Assume that $\sigma \leq \frac{\epsilon p}{\ell}$ for some small $\epsilon > 0$.

1. Derive a lower bound for the probability

$$\Pr\left[e_i \leq \frac{p}{4\ell} \forall i = 1, \ldots, \ell\right]$$

using the facts given above and a union bound.

2. What does this lower bound mean for the probability of decryption failure, i.e. for the probability that encrypting a message m and decrypting it again returns $m' \neq m$?

Note: Our analysis is quite loose here, it suffices to pick $\sigma = \frac{\epsilon q}{\sqrt{\ell}}$, but it is slightly harder to show that.

Exercise 2. (**Broken variants of Regev's encryption scheme**)

In this exercise, you will find attacks against insecure modifications of Regev's encryption scheme. For each attack, give a detailed argument why it works.

1. Describe how to break Regev's encryption scheme for $\sigma = 0$, i.e. in the case where it always holds that $e_i = 0$ for all i. More precisely, describe an algorithm that given a public key pk and a ciphertext $c = \mathsf{Enc}_{pk}(m)$, recovers the plaintext m.
2. Describe how to break Regev's encryption scheme where instead of picking the r_i at random, $r_i = 1$ for all i.
3. Let σ be chosen such that $\Pr_{x \leftarrow q_\sigma}[x \neq 0 \bmod p] = \frac{1}{2\ell}$. Describe how to break Regev's encryption scheme in that case.

Exercise 3. (Towards homomorphic encryption from LWE)

Consider the following symmetric-key encryption scheme for messages $m \in \{0, 1, ..., \ell - 1\}$.

- Key generation: choose uniform $\mathbf{s} \in \mathbb{Z}_p^n$.
- Encryption of message $m \in \{0, 1, \dots \ell - 1\}$: Choose uniform $c_0 = \mathbf{a} \in \mathbb{Z}_p^n$ and $e \leftarrow q_\sigma$. Output (c_0, c_1) with $c_1 = \mathbf{a}^\top \cdot \mathbf{s} + e + \left\lfloor \frac{m(p-1)}{\ell} \right\rceil$.

1. Describe a decryption algorithm that gets as input a secret key $\mathbf{s}$ and a ciphertext $c = (c_0, c_1)$ and outputs a plaintext, and show that your decryption algorithm is correct if σ is small enough.
2. Let $c = (c_0, c_1)$ and $c' = (c'_0, c'_1)$ be ciphertexts obtained by encrypting messages m and m'. consider $c'' = (c''_0, c''_1)$ given by $c''_i = c_i + c'_i$ for $i = 0, 1$. Apply the decryption algorithm you described in 1) to c''. Under which condition do you get $m + m' \bmod \ell$?

Problem sheet 9 for Course 02231, 2025

These practice problems have the purpose of helping you understand the material better and learning the skills that are necessary to analyze cryptographic constructions, and sometimes to prepare you for the next class. All answers should be supported by a written justification. To gauge whether a justification is sufficient, ask yourself if your peers would be convinced by it without additional explanations.

Exercise 1. (Secure digital signatures for messages of arbitrary length)

In the lecture you saw a proof sketch how RSA signatures for arbitrary-length messages are constructed. Because plain RSA signatures are not secure, not even for fixed-length messages, we had to use the Random Oracle Model (ROM). As it turns out, the ROM is not necessary when using a hash function and a secure digital signature scheme (DSS) to construct a DSS for arbitrary-length messages. Assume that $\Sigma = (\Sigma.\mathsf{KeyGen}, \Sigma.\mathsf{Sign}, \Sigma.\mathsf{Ver})$ is a secure DSS for messages of length n bits. Furthermore, assume that H is a collision-resistant hash function mapping $\{0,1\}^*$ to $\{0,1\}^n$.

1. Construct a signature scheme which can sign messages from $\{0,1\}^*$, using the given signature scheme and H, following the idea of RSA-FDH.
2. Show that your constructed signature scheme is EUF-CMA secure, given that Σ is secure and H is collision-resistant.

Exercise 2. (Non-Repudiation)

In the lecture, it was mentioned that secure DSS have the non-repudiation property. Here, we explore this property a bit more explicitly.

1. Show that the following holds: Assume that a signer uses a digital signature scheme Σ, has generated a public key vk for it but never releases the signing key sk to anyone. Furthermore, assume that there exists an efficient algorithm $\mathcal{A}$ which generates a valid signature verifiable under vk that was never generated by the signer (i.e. the signature scheme does not provide non-repudiation). Then $\mathcal{A}$ can be used to efficiently distinguish $L_{sig-real}$ and $L_{sig-fake}$ for the digital signature scheme.
2. Why does the same argument not apply for a single message sender in a MAC scheme, even though it's security is essentially defined in the same way?

Exercise 3. (**Why we need hashes for RSA signatures**)

It might be tempting to design RSA signatures without using a cryptographic hash function H. The idea could be the following, alternative signing algorithm:

1. The input is a message $m \in \mathbb{Z}_N^*$ such that $m < \lfloor N/2^{16} \rfloor$ (i.e. the first two bytes of m are 00).
2. We then sign this m by computing $\sigma = m^d \bmod N$.

For verification, we now do the following:

1. Check that the message $m \in \mathbb{Z}_N^*$ is of the correct form, namely that it is smaller than $\lfloor N/2^{16} \rfloor$.
2. Check that $\sigma^e = m \bmod N$.

Let us assume that raising to the eth power modulo N is a perfect permutation, i.e. on input $x \in \mathbb{Z}_N^*$ the value $x^e \bmod N$ is uniformly random in $\mathbb{Z}_N^*$. Show that an attacker, by only using the public key pk, can break the EUF-CMA property for the aforementioned "signature" scheme by making around 2^{16} exponentiations modulo N.

Exercise 4. (**Hashing to $\mathbb{Z}_N^*$**)

In the lecture, we assumed that the hash function H used in the RSA-FDH construction outputs a value in $\mathbb{Z}_N^*$ for every message m. It might appear unrealistic to construct such a hash function, because this is a very specific requirement on the output set.

Show that using a hash function $\hat{H}$ which outputs random values from $\mathbb{Z}_N$ instead of $\mathbb{Z}_N^*$ is sufficient for the task. To do so, assume that p, q are both of size around 2^{1000} (i.e. 1000 bits long), and calculate how many outputs of $\hat{H}$ will lie outside of $\mathbb{Z}_N^*$ by not being coprime to N. To do so, you can count the total number of integers in $[0, N-1]$ that are not coprime to N in a clever way.

Bonus: Assume that $\hat{H}$ is outputting values in $\mathbb{Z}_N \setminus (\mathbb{Z}_N^* \cup \{0\})$ with a high probability over its possible inputs. Show that you can use it to factor N efficiently with essentially the same probability.

Problem sheet 10 for Course 02231, 2025

These practice problems have the purpose of helping you understand the material better and learning the skills that are necessary to analyze cryptographic constructions, and sometimes to prepare you for the next class. All answers should be supported by a written justification. To gauge whether a justification is sufficient, ask yourself if your peers would be convinced by it without additional explanations.

Exercise 1. (Computing Discrete Logarithms with smooth group order)

Let p be a prime and $g, h \in \mathbb{Z}_p^*$ where g has order $p-1$. This means that $g^{p-1} = 1 \bmod p$ while $g^x \neq 1 \bmod p$ for every $1 \leq x < p-1$.
Given p, g, h the discrete logarithm problem is to find the unique $a \in \mathbb{Z}_{p-1}$ such that $g^a = h \bmod p$.

For this exercise, let $p = 43$, $g = 12, h = 5$.

1. Try out all possible choices of a (e.g. using a Python script or Excel) to find the discrete logarithm. For an arbitrary p, how many multiplications modulo p would you have to do (in the worst case) to find a this way?
2. We observe that $p - 1 = 2 \cdot 3 \cdot 7$ and want to use this to simplify the computation of the discrete logarithm. Let $x = (p-1)/2, y = (p-1)/3, z = (p-1)/7$ and consider the elements g^x, g^y, g^z. What do you know about the order of these elements modulo p?
3. We can find the value $a \bmod 2$ by computing the discrete logarithm of h^x for the base g^x. Similarly, we can obtain $a \bmod 3$ from g^y, h^y and $a \bmod 7$ from g^z, h^z. Can you use this to find $a \in \mathbb{Z}_{42}$ more efficiently?
4. More generally, assume that $p - 1$ has ℓ prime factors that are all smaller than B. Can you (roughly) say how many multiplications modulo p you have to do, in comparison to the trivial method that tries out all choices of a, to recover the discrete logarithm?
5. Given the results of this exercise, what can you say about the security of the discrete logarithm in groups of smooth order (i.e. where the order of the group only has small prime factors)?

Exercise 2. (When the Decisional Diffie Hellman Problem is easy)

Let p be a prime. In the lecture, we considered the Decision Diffie Hellman (DDH) problem in the case when $g \in \mathbb{Z}_p^*$ was of large prime order q such that $q | p-1$. Now instead, assume that $g \in \mathbb{Z}_p^*$ is a generator of the whole group $\mathbb{Z}_p^*$.

Show that, in this case, the DDH problem is easy: one can distinguish tuples of the form $(g, g^a, g^b, g^{a \cdot b})$ for $a, b \in \mathbb{Z}_{p-1}$ from tuples of the form (g, g^a, g^b, g^c) for $a, b, c \in \mathbb{Z}_{p-1}$ with a very good chance. For this, use the observations from the previous exercise and consider what happens if you raise each element in the tuple to $(p-1)/2$.

Exercise 3. (**From Diffie Hellman to Public-Key Encryption**)

Let p be a prime and $g \in \mathbb{Z}_p^*$ be of large prime order $q|p-1$. In the Diffie Hellman Key Exchange Protocol, Alice and Bob exchange messages $A = g^a \bmod p, B = g^b \bmod p$ where $a, b \in \mathbb{Z}_q$.

1. Assume that Bob publishes the message B as a public key, while he keeps b as his secret key. Alice now encrypts a message $m \in \{0,1\}$ as follows:
   (a) She chooses $a \in \mathbb{Z}_q, r \in \mathbb{Z}_q^*$, and computes $c_1 = g^a \bmod p$.
   (b) If $m = 0$ then she sets $c_2 = B^a \bmod p$, otherwise she sets $c_2 = B^a \cdot g^r \bmod p$.
   (c) She lets c_1, c_2 be the ciphertext for Bob.

   Show how Bob can recover the message.
2. Show that this encryption scheme is IND-CPA secure assuming DDH is hard in the group $\mathbb{Z}_p^*$ with generator g. Namely, show that if there exists an attacker that wins the IND-CPA security game with probability $P > 1/2$, then we can use it to construct an algorithm that breaks DDH with the same probability.

Exercise 4. (**Bad groups for Diffie-Hellman**)

Explain in detail why the following groups are not suitable for use in the Diffie-Hellman key exchange protocol. Below, let p be a prime and let $(S_n, \circ)$ be the group of permutations of n objects ($\circ$ denotes composition of permutations).

1. $(\mathbb{Z}_p, +)$, i.e. the group of remainders modulo p with addition as group operation.
2. $\left(\left\{\begin{pmatrix} 1 & a \\ 0 & 1 \end{pmatrix} \middle| a \in \mathbb{Z}_p\right\}, \cdot\right)$ where $\cdot$ denotes matrix multiplication modulo p.
3. $(\langle g \rangle, \circ)$ for a full cycle $g \in S_n$[a]

[a] A full cycle is a permutation $g \in S_n$ where $\{1, g(1), g^2(1), \ldots\} = \{1, 2, 3, \ldots, n\}$ (as sets).

Problem sheet 11 for Course 02231, 2025

These practice problems have the purpose of helping you understand the material better and learning the skills that are necessary to analyze cryptographic constructions, and sometimes to prepare you for the next class. All answers should be supported by a written justification. To gauge whether a justification is sufficient, ask yourself if your peers would be convinced by it without additional explanations.

Exercise 1. (An elliptic curve group)

Let p be a prime and $a, b \in \mathbb{Z}_p$. Define the set

$$G = \{(x, y) | x, y \in \mathbb{Z}_p, y^2 = x^3 + a \cdot x + b\} \cup \{O\}.$$

O is sometimes called "the point at infinity". We define the group law as follows. For all $P, Q \in G$, $P = (x_1, y_1)$, $Q = (x_2, y_2)$,

- $O + P = P + O = P$.
- $-P = (x, -y)$
- $P - P = O$
- If $Q \neq -P$, set

$$\lambda = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1} & \text{if } x_1 \neq x_2 \\ \frac{3x_1^2 + a}{2y_1} & \text{if } x_1 = x_2 \text{ and } y_1 \neq 0. \end{cases}$$

  and define $R = P + Q$, $R = (x_3, y_3)$ with $x_3 = \lambda^2 - x_1 - x_2$ and $y_3 = (x_1 - x_3)\lambda - y_1$.

Note: Here all computations are mod p, so $\frac{s}{t} = s \cdot t^{-1}$ where t^{-1} is the inverse of t mod p.

1. For $p = 11$, $a = 2$, $b = 7$, compute $P = (6, 2) + (7, 1)$, $Q = -(7, 1)$ and $R = P + Q$.
2. (Bonus exercise) Show that the defined group addition is commutative, i.e. show that for all $P, Q \in G$ it holds that $P + Q = Q + P$.

Exercise 2. (X3DH)

Consider the X3DH protocol as introduced in the lecture, without the optional 4th Diffie-Hellman exchange which uses $k_4 = DH(EK_A, EK_B)$.

1. For each of the three intermediate keys k_i, $i = 1, 2, 3$, describe what security property is lost if key k_i is omitted from the final key derivation. Example for $i = 1$: in what way is the protocol insecure if we compute the final key as $k = H(k_2, k_3)$?

Exercise 3. (**Schnorr signatures**)

In the lecture we learned about RSA-FDH signatures. There are also signature schemes which rely on the DLOG problem, one of which is called Schnorr's signature scheme.
As before, let p be a prime and $g \in \mathbb{Z}_p$ of order q. Furthermore, let $H : \{0,1\}^* \to \mathbb{Z}_q$ be a collision-resistant hash function. The Schnorr signature scheme consists of the following 3 algorithms:

$Keygen$

1. Sample $s \leftarrow \mathbb{Z}_q$ uniformly at random.
2. Compute $S = g^s \bmod p$.
3. Output $(pk, sk) = (S, s)$.

$Sign$ on input $m \in \{0,1\}^*$ and sk:

- Sample $a \leftarrow \mathbb{Z}_q$ uniformly at random.
- Compute $A = g^a \bmod p$.
- Compute $e = H(A, m)$ and $z = a + e \cdot s \bmod q$.
- Output $\sigma = (A, z)$.

$Verify$ on input $pk = S$, $m \in \{0,1\}^*$ and $\sigma = (A, z)$:

- Compute $e = H(A, m)$.
- Output 1 if $A \cdot S^e = g^z \bmod p$, otherwise output 0.

1. Verify that the Schnorr signature scheme is correct. For that, imagine that pk, sk is generated correctly and σ is a correctly generated signature on m. Then $Verify(pk, m, \sigma) = 1$ should always hold.

2. Assume that there exists an algorithm $\mathcal{A}$ which, on input $g, A \in \mathbb{Z}_p$ outputs $a \in \mathbb{Z}_q$ such that $g^a = A \bmod p$. How can you use $\mathcal{A}$ on a Schnorr public key pk to generate signatures on arbitrary messages that are valid for pk?

Exercise 4. (**DDH and key reuse**)

For both the El Gamal encryption scheme and for X3DH, it is important that Diffie-Hellman remains secure even if one of the private exponents has been used before. For $g \in \mathbb{Z}_p$ of order q, consider therefore the following variant of the DDH problem:
DDH': *Sample* $a, b, c, d, e \in \mathbb{Z}_q$ *uniformly random and independent. The attacker must distinguish* $(g, g^a, g^b, g^c, g^d, g^e)$ *from* $(g, g^a, g^b, g^c, g^{ab}, g^{ac})$.
We can write this as two libraries, similar to $L_{ddh-real}$ and $L_{ddh-ideal}$ from the lecture, as follows:

$L_{ddhp-ideal}$**:**

- Initially, sample $a, b, c, d, e \in \mathbb{Z}_q$ uniformly random and independent.
- The function $GetInstance()$ outputs $(g, g^a, g^b, g^c, g^d, g^e)$.

$L_{ddhp-real}$**:**

- Initially, sample $a, b, c \in \mathbb{Z}_q$ uniformly random and independent.
- The function $GetInstance()$ outputs $(g, g^a, g^b, g^c, g^{ab}, g^{ac})$.

In this exercise, you will show that any attack on DDH' can be translated into an attack on DDH.

1. Let $\mathcal{A}$ be an algorithm that distinguishes $L_{ddhp-real}$ from $L_{ddhp-ideal}$. What in the behavior of both libraries is different, so that $\mathcal{A}$ can distinguish it?

2. We want to construct an algorithm $\mathcal{B}$ that uses $\mathcal{A}$ as a sub-routine, so that $\mathcal{B}$ solves DDH. The algorithm $\mathcal{B}$ would interact either with $L_{ddh-real}$ or $L_{ddh-ideal}$ and has to tell which one it is. It therefore locally simulates a library (based on the output of the library it talks to) and lets $\mathcal{A}$ talk to the simulated library. Can you describe this process? In your simulation, you should simulate $L_{ddhp-real}$ when $\mathcal{B}$ talks to $L_{ddh-real}$ and simulate $L_{ddhp-ideal}$ when you talk to $L_{ddh-ideal}$.

3. Based on your simulation, define how $\mathcal{B}$ uses the output of $\mathcal{A}$ to tell the two libraries apart. How successful will $\mathcal{B}$ be, if $\mathcal{A}$ is correct α percent of the time?

Exercise 5. (**The Pedersen Commitment**)

Commitments are an advanced cryptographic primitive. They allow a sender to "commit" to a message m towards the receiver by sending a value c. Having only c (i.e. before m is "opened" to the receiver), the receiver cannot say what message m is contained inside c. At the same time, once c is sent to the receiver then the sender cannot change his mind and open c to another message m' anymore towards the sender. More formally, a commitment scheme consists of two algorithms:

Commit A Com algorithm which, on input m outputs values c, d.

Open An $Open$ algorithm which, on input m, c, d outputs a bit.

It is required that the commitment scheme is binding and hiding:

Binding It should be computationally difficult for a sender to generate values m, m', d, d', c such that $Open(m, c, d) = Open(m', c, d') = 1$ while $m \neq m'$. Note that sender has a free choice of all these values, as long as both messages m, m' are different but open the same commitment c which the sender can also choose.

Hiding Given m_0, m_1 by an adversary, this adversary should not be able to decide if it is a commitment to m_0 or m_1 for an honestly generated commitment c (similar to the IND-CPA property for encryption schemes, where the adversary can pick two "potential" messages but cannot say which one is ultimately encrypted in the ciphertext).

Towards constructing a commitment scheme, let us, as before, assume that p is a prime and $g \in \mathbb{Z}_p^*$ is of large prime order $q|p-1$. We assume that p, q, g are public knowledge for everyone. A first attempt for a commitment scheme is the following:

Commit On input $m \in \mathbb{Z}_q$, output $c = g^m \bmod p$ and $d = \perp$.

Open On input $m \in \mathbb{Z}_q, c \in \mathbb{Z}_p^*$ output 1 if $c = g^m \bmod p$ and 0 otherwise.

Show that this construction is insecure because it is not hiding!

A version of this, which bears resemblance to the previous exercises, is actually secure! It is called the *Pedersen Commitment* and it works as follows, assuming an additional $h \in \langle g \rangle$ (i.e. $h = g^a$ for some value a). h is also of prime order q and also publicly known (and fixed for sender and receiver):

Commit On input $m \in \mathbb{Z}_q$, sample a random $r \in \mathbb{Z}_q$ and output $c = g^m h^r \bmod p$ and $d = r$. The receiver will obtain c while the sender keeps d to itself.

Open On input $m \in \mathbb{Z}_q, c \in \mathbb{Z}_p^*, d \in \mathbb{Z}_q$ output 1 if $c = g^m h^r \bmod p$, otherwise output 0.

1. Assume that neither sender nor receiver know the discrete logarithm of h to the base g modulo p. Then the aforementioned commitment scheme is binding. To prove this, assume for contradiction that there exists a sender algorithm that can, on input p, q, g, h, generate values m, m', c, r, r' such that $g^m h^r \bmod p = g^{m'} h^{r'} \bmod p$ where $m \neq m'$. Then show that you can use this sender algorithm to compute the discrete logarithm of h to base g modulo p!

2. Show that the commitment scheme is also hiding! You can show that an honestly generated commitment $c = g^m h^r \bmod p$ could have been generated by any other message m' using a different randomness r'.

Problem sheet 12 for Course 01410, 2025

Exercise 1. (**Period finding for breaking Diffie Hellman**)

Recall the discrete logarithm problem: given $p \in \mathbb{N}$ and $g, h \in \mathbb{Z}_p^*$, find $a \in \mathbb{N}$ such that $g^a = h \bmod p$. It turns out that the discrete logarithm problem can also be solved via period finding. In this case, however, a two-dimensional version of period finding is needed. This 2D period finding problem is as follows: For a function $f : \mathbb{N}^2 \to \mathbb{N}$, find $\alpha, \beta \in \mathbb{N}$ such that $f(x + \alpha, y + \beta) = f(x, y)$ for all $x, y \in \mathbb{N}$. For the discrete logarithm problem as defined above, define

$$f(x, y) = g^x h^{-y} \bmod p.$$

Show that the function is periodic. Given a period (α, β), how can you compute the discrete logarithm?

Exercise 2. (**When will we have a cryptographically relevant quantum computer?**)

Take a look at the 2024 Quantum Threat Timeline Report from the Global Risk Institute, available here:
`https://globalriskinstitute.org/publication/2024-quantum-threat-timeline-report/`
When do we need to expect RSA and Diffie-Hellman as we use it today to be broken? What are the implications, when do you think do we need to switch to using post-quantum cryptography?