

## Round 1: One-way functions (OWF)

### Protocol repetition

1 of 2 ✕

By repeating, the verifier (Kirthi) can recover the entire coloring if the verifier chooses a different edge each time.

✕ ☒ True

✓ ☐ False

#### Explanation

The prover permutes the colors randomly each time, so that each time, the verifier only sees an edge with two random colors. In particular, the colors which the verifier sees might not be compatible with the colors seen in previous round. To see this, compare the result of round 1 with the result of round 3 (coming next in the video).

### Protocol repetition

2 of 2 ✕

Why do we need to repeat the protocol? Click all that are correct (can be 0, 1 or 2 answers).

☐ By repeating, the verifier (Kirthi) can learn the entire coloring.

✓ ☒ If the graph is not 3-colorable, in each round, there must be at least 1 edge where the prover put the same color on both ends. The verifier only chooses each edge with probability  $1/\text{number of edges}$ , so we need to repeat to increase the probability of catching a cheating prover.

✓ ☐ If the prover does not know a 3-coloring and just colors all nodes blue and one node black, then the verification might still pass with probability  $\text{nbr-of-edges-adjacent-to-the-black-node}/\text{nbr-of-edges}$ .

#### Explanation

Answer 1 is incorrect (see explanation of previous question).

### Deterministic hash-functions

1 of 2 ✕

Sha-3 is a (deterministic) hash-function. Someone chooses a bitstring  $a$  encoding the color "blue" and computes  $y_1 \leftarrow \text{Sha-3}(a)$  and  $y_2 \leftarrow \text{Sha-3}(a)$ . Which of the following is true?

✓ ☒  $y_1$  and  $y_2$  are equal.

☐  $y_1$  and  $y_2$  are different.

#### Explanation

$y_1$  and  $y_2$  are equal, because Sha-3 is deterministic.

## Deterministic hash-functions


2 of 2 ✕

I am given a value  $y$  and I am told that  $y$  was computed as  $\text{Sha-3}(a)$ , where  $a$  is a 5-bit-long string. How expensive is it to recover string  $a$  from  $y$ ?

- ✓ ☐ Roughly  $32 \times \text{time-to-evaluate-sha-3-on-a-5-bit-input}$ .
- ✗ ☒ It is infeasible because Sha-3 is a one-way function.

### Explanation

Answer 1 is correct. The attack proceeds via brute force. One tries out all possible 5-bit-inputs. There are  $2^5=32$  such inputs  $x$  and for each of them one checks whether the  $\text{Sha-3}(x)=y$ . Surprisingly, one can recover the input, although Sha-3 is a one-way function! What does this mean? (We'll see the answer in the rest of the lecture.)

$\text{Exp}_{f,A}^{\text{OW}}(1^\lambda)$  *long*  *function f*  
*bitstring*  
 $x \leftarrow \{0, 1\}^\lambda$  *adversary A*  
 $y \leftarrow f(x)$   
 $x' \leftarrow \mathcal{A}(1^\lambda, y)$  *x' maps to y*  
 if  $f(x') = y$  then  
     return 1 *successful*  
     return 0 *unsuccessful*

## One-wayness experiment

1 of 1 ✕

Click all answers which you think are correct (0, 1, 2 or 3).

- ☐ The only way for the adversary to win the experiment is output  $x'=x$ .
- ✓ ☒ The adversary can output any pre-image  $x'$  of  $y$ .
- ✓ ☒ The adversary receives a value  $y$ , where  $y$  is computed by first sampling a uniformly random bitstring  $x$  of length  $\lambda$  and then applying  $f$  to  $x$ .

### Explanation

Answer 1 is not correct. The adversary  $A$  can output \*any\*  $x'$  such that  $f(x')=y$ . Of course,  $x'=x$  is one possibility, but if  $f$  is not injective, there are also other possible pre-images which the adversary  $A$  can output. Therefore, answer 2 is correct. Answer 3 describes the experiment well in words.

## Apparent contradiction?

1 of 1 ✕

Given this definition, how can we resolve the apparent contradiction between Sha-3 being a one-way function and the possibility to invert Sha-3 efficiently if we know that input is one of the strings "red", "blue" or "black"? (Click all which are true. Can be 0, 1 or 2 answers.)

- ✓ ☒ Any efficiently computable, deterministic function can be inverted when the inputs are chosen from a small set. This point is unrelated to whether the function is a one-way function or not.
- ✓ ☒ In the security experiment, the input  $x$  to the function is sampled as a long, uniform input of length  $\lambda$ . As  $\lambda$  is a (big) parameter, the security definition of one-wayness only says that inverting on uniform inputs of length  $\lambda$  is hard (whereas it might be easier on short values).

### Explanation

Answer 1 is correct because trying out all inputs from the small set and evaluating the function on it is an efficient inversion strategy when (1) the deterministic function is efficiently computable and (2) the set is indeed small.

Answer 2 is correct because indeed,  $x$  is sampled uniformly at random in the experiment.

## Definition of $g$

1 of 1 ✕

Why is  $y_1$  assigned but not used in the code of  $g$ ?

- ✗ ☐ The first line assigns only the first bit of  $y$  and the second line assigns the other bits of  $y$ .
- ✓ ☒ This is a typo. The code should have been  $y \leftarrow y_1 \parallel 0\dots 0$  instead of  $y \leftarrow f(x) \parallel 0\dots 0$ . (The result is the same, but it is strange to make an assignment and then ignore it.)

### Explanation

(I just added this quiz to point out that there is a typo so you don't try to find deep meaning in such a strange code!)

## Transforming one-way functions and attackers

1 of 1 ✕

Which of the following is true (several answers might be correct).

- ✓ ☒ We build a one-way function  $g$  from a one-way function  $f$ .
- ☐ We build a one-way function  $f$  from a one-way function  $g$ .
- ☐ We build an attacker  $A_g$  against  $g$  from an attacker  $A_f$  against  $f$ .
- ✓ ☒ We build an attacker  $A_f$  against  $f$  from an attacker  $A_g$  against  $g$ .

### Explanation

Notice the asymmetry. To show that one-wayness of  $f$  implies one-wayness of  $g$ , we need to show that if  $g$  is broken, then  $f$  is broken. Therefore, we transform an adversary against  $g$  into an adversary against  $f$ .

## Round 2: Pseudorandom generator (PRG)

### Quiz: One-Way Function Definition Recap

1 of 1 ✕

An (efficient) adversary  $A$  for a one-way function  $f$  is given  $y$  (which was computed as  $f(x)$  for a random  $x$ ) and the security parameter and is considered successful if it succeeds (with high enough probability) to return a pre-image  $x'$  such that  $f(x')=y$ . Which of the following two statements is true?

- ✗ ☒ The adversary  $A$  is only considered successful if  $x'=x$ .
- ✓ ☐ The adversary  $A$  is also considered successful if  $x'$  is not equal to  $x$  (as long as  $f(x)=y$ ).

#### Explanation

The security game only checks whether  $f(x')=y$ . When  $f$  is not injective, then  $f(x')=y$  does not necessarily mean that  $x=x'$ . See next quiz for a nice example :-)

### Quiz: Constant OWF?

1 of 1 ✕

Let  $f$  be an efficiently computable function such that for all inputs  $x$ ,  $f$  returns the 128-bit string  $0^{128}$ .

- ☐  $f$  is a one-way function.
- ✓ ☒  $f$  is not a one-way function.
- ☐ We do not know whether  $f$  is a one-way function.

#### Explanation

$f$  is not a one-way function since any possible input is a valid pre-image (see previous quiz). For example, consider the following efficient adversary  $A$  which on input  $y$  and the security parameter  $1^n$  returns  $0^n$ . This adversary inverts  $f$  with probability 1 (for all possible input lengths).

### Quiz: Image of a PRG

1 of 1 ✕

We have a pseudorandom generator with stretch  $s(n)=n$ , i.e., for all  $x$  in  $\{0,1\}^n$ , we have that  $|g(x)|=n+n=2n$ . Define the Image of  $g$  as

$\text{Im}(g) := \{y \text{ in } \{0,1\}^{2n} \text{ such that there exists } x \text{ in } \{0,1\}^n \text{ which yields } g(x)=y\}$ .

If we draw a uniformly random value  $y$  from  $\{0,1\}^{2n}$ , what is the probability that  $y$  is in  $\text{Im}(g)$  ? Check all answers which are true (can be none, one, two, three, four of five answers). The probability is...

☐  $> 1/2$

☐  $> 1/n$

☐  $1/n^2$

☐  $> 2^{n-n}$

✓ ☒ less or equal to  $2^{n-n}$

#### Explanation

$\text{Im}(g)$  is at most of size  $2^n$ , because it contains at most 1 value  $y$  per each  $x$  in  $\{0,1\}^n$ . ( $\text{Im}(g)$  might be even smaller than  $2^n$  if two different  $x$  and  $x'$  are mapped to the same value  $y=g(x)=g(x')$ ). Thus,  $2^n$  values or less out of  $2^{2n}$  are in the image of  $g$ . If we choose each value in  $\{0,1\}^{2n}$  with the same probability, then the probability is less or equal to  $2^n/2^{2n}=2^{n-2n}$ .

### Quiz: Is a PRG deterministic?

1 of 1 ✕

A pseudorandom generator is a deterministic function.

✓ ☒ True

☐ False

#### Explanation

A pseudorandom generator deterministic, because it models the transformation of some amount of random bits into a larger amount of pseudorandom bits. If this transformation is allowed to be randomized, then the task of the pseudorandom generator becomes trivial: It simply samples as many uniformly random bits as desired.

### Quiz: PRG Definition

1 of 1 ✕

Consider the following modification of the ideal PRG security experiment. Namely, consider that the ideal PRG experiment draws  $y$  uniformly at random from  $\{0,1\}^n$  rather than from  $\{0,1\}^{n+s(n)}$ . Is the following statement true or false?

There is a polynomial-time adversary which can distinguish between the real PRG experiment and this modified ideal PRG experiment.

✓ ☐ True

✗ ☒ False

#### Explanation

If the ideal experiment draws  $y$  from  $\{0,1\}^n$ , then in the ideal experiment, the input  $y$  to the adversary is shorter than in the real experiment (where the length of  $y$  is  $n+s(n)$  which is strictly greater than  $n$  as  $s(n)$  is at least 1). Thus, an adversary can distinguish between the two experiments by checking whether  $y$  is of length  $n+s(n)$  (real game) or of length  $n$  (ideal game).

## Quiz: Negligible functions

1 of 1 ✕

The following functions  $\nu_1, \dots, \nu_5$  take a natural number  $n$  as input and return a positive real number. Check the boxes of all functions which are negligible. (Can be 0, 1, 2, 3 or 4 functions.)

Hint: To decide this question, you can use the definition of negligible and/or check whether property 2 holds, i.e.,  $p(\lambda) \cdot \nu_i(\lambda)$  is negligible for all polynomials  $p$ . Especially for those which are not negligible, property 2 is convenient. Note that the constant 1 function is not negligible.

✓ ☒  $\nu_1(n) := 1/2^n$

☐  $\nu_2(n) := 1/n$

☐  $\nu_3(n) := 1/\log n$

✓ ☒  $\nu_4(n) := 1/2^{\{n/2\}}$

☐  $\nu_5(n) := 1/1000000000000000$

### Explanation

- $1/2^n$  is exponentially small and exponential functions grow faster than any polynomial. Thus, the definition applies to  $\nu_1$ .
- $n$  is a polynomial, and  $n \cdot \nu_2(n) = 1$  is not negligible, thus, property 2 does not hold for  $\nu_2$ .
- $n$  is a polynomial, and  $n \cdot \nu_3(n) > n \cdot \nu_2(n) = 1$  is not negligible, thus, property 2 does not hold for  $\nu_3$ .
- $1/2^{\{n/2\}}$  is exponentially small and exponential functions grow faster than any polynomial. Thus, the definition applies to  $\nu_4$ .
- $\nu_5(n)$  is a constant. A constant does not tend to zero. In particular, choosing an adequate constant  $c$  (A constant is also a polynomial), we obtain  $\nu_5(n) \cdot c = 1$  which is not negligible. Thus, property 2 is not satisfied for  $\nu_5$ .

## Quiz: Hardcore Bits

1 of 1 ✕

Let  $f$  be a one-way function, and write its input as two halves  $x = x_{\text{left}} || x_{\text{right}}$ . Consider the one-way function  $h(x) := f(x_{\text{left}}) || x_{\text{right}}$ .

Is the following statement true or false?

The function  $b: \{0,1\}^* \rightarrow \{0,1\}$  defined as  $b(x) := \text{last-bit-of-}x$  is a hardcore bit for  $h$ ?

✕ ☒ True

✓ ☐ False

### Explanation

False, because the adversary can compare the last bit of  $h(x)$  with the hardcore bit  $z$ . In the real hardcore bit experiment, those two bits are always equal. In turn, in the ideal hardcore bit experiment, the bit  $z$  is drawn uniformly at random, and thus, the last bit of  $h(x)$  and  $z$  are different with probability  $1/2$ .

More details:

An adversary can distinguish between the real and ideal game by returning 0 when the two bits are equal and 1, else. It then returns 1 with probability 1 in the real experiment, but only with probability  $1/2$  in the ideal experiment and thus has very high distinguishing advantage (in particular,  $1/2$  is not a negligible function).

## Round 3: Pseudorandom functions (PRF)

### Proof by contradiction

2 of 2 ✕

We used the "proof by contradiction" argument in the proof in order to prove an implication. Which of the following statements captures the process of "proof by contradiction" best?

- ✓ ☒ We want to show that statement X implies statement Y. We prove this by assuming that Y is wrong and showing that then, X must be wrong, too. Thus, if X is true, then Y must be true as well, which proves that statement X implies statement Y.
- ☐ We want to show that statement X implies statement Y. We prove this by assuming that X is wrong and showing that then, Y must be wrong, too. Thus, if Y is true, then X must be true as well, which proves that statement X implies statement Y.

#### Explanation

The easiest way to think about this is, perhaps, in terms of a truth table.  $X \Rightarrow Y$  is true in the following cases:

X true, Y true

X false, Y true

X false, Y false

So, the only "bad case" which can happen is that X is true while Y is actually false! So, if we can exclude this case, then we are done with the proof. This is what a proof by contradiction achieves.

### Proof by contradiction

1 of 2 ✕

Check all which are true (can be 0, 1, 2, 3, 4, 5 or 6 answers). For the proof of this theorem, we had to...

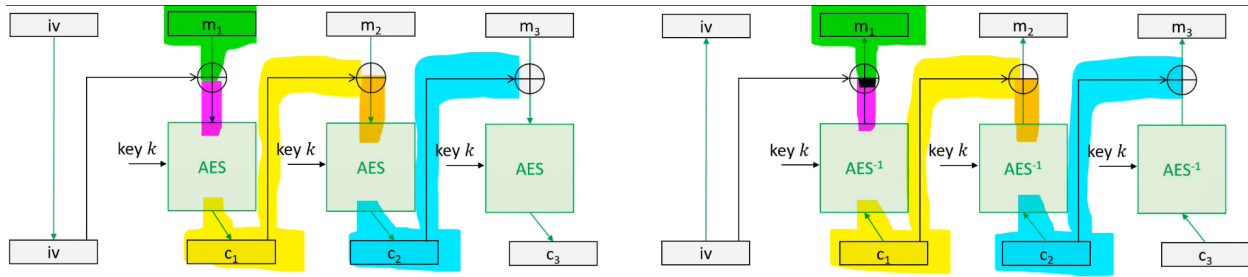
- ✓ ☒ ...construct a function  $f'$  based on a function  $f$ .
- ✓ ☐ ...prove that  $f'$  is efficient (i.e., polynomial-time), assuming that  $f$  is efficient (i.e., polynomial-time).
- ✓ ☒ ...describe an adversary  $B$  based on an adversary  $A$ .
- ✓ ☒ ...prove that adversary  $B$  is efficient (i.e., probabilistic polynomial-time), assuming that  $A$  is efficient (i.e., probabilistic polynomial-time).
- ☐ ...prove that if  $A$  has non-negligible probability in inverting  $f$ , then  $B$  has non-negligible probability in inverting  $f'$ .
- ✓ ☒ ...prove that if  $A$  has non-negligible probability in inverting  $f'$ , then  $B$  has non-negligible probability in inverting  $f$ .

#### Explanation

A one-way function needs to be efficiently computable (\*), thus we have to ensure that  $f'$  is efficiently computable. An adversary can only be considered to break the one-wayness property if the adversary is efficient (probabilistic polynomial-time) (\*\*), thus, we need to ensure that  $B$  is efficient.  $A$  was defined to be an inverter (adversary) against  $f'$  and was assumed to have non-negligible inversion probability against  $f'$ .  $B$  is a candidate inverter against our OWF  $f$  and thus, we have to show that  $B$  has non-negligible inversion probability against  $f$ . (In fact, it is not at all clear what  $B$  would do with an input from  $f'$ , since it expects an input from  $f$ ).

(\*) This follows from the definition of OWF, but here is also an intuitive argument: A function which is very hard might be hard in both directions, so conceptually, the term one-way function would be a bit of a misnomer in the case where the forward direction is hard, too.

(\*\*) This follows from the definition of OWF - the reason that it makes sense that the adversary needs to be efficient is that an inefficient adversary can always invert the OWF simply by going through all the possible candidate pre-images.



## CBC mode

1 of 1 ×

Which of the following is true (only 1 answer):

- ✓ ☒  $c_1 \text{ xor } \text{AES}^{-1}(k, c_2)$  is equal to  $m_2$ , because  $c_2$  is equal to  $\text{AES}(k, c_1 \text{ xor } m_2)$  and because for all values  $x$ , we have that if  $y = \text{AES}(k, x)$ , then  $\text{AES}^{-1}(k, y) = x$ .
- ☐  $m_2$  is equal to  $c_1 \text{ xor } m_1$ .
- ☐  $m_3$  is equal to  $\text{AES}^{-1}(k, c_3)$

### Explanation

$m_1$  and  $c_1$  are not xored at any point in the diagram.  $m_3$  needs to be xored with  $c_2$  to obtain  $\text{AES}^{-1}(k, c_3)$ .

## Quiz: GGM

1 of 1 ×

On how many input can I evaluate  $f_{\text{GGM}}(k, \cdot)$  if I fix input length of inputs  $x$  to length 10?

- ☐ 10
- ☐ 20
- ✓ ☒  $2^{10}$

### Explanation

There are  $2^{10}$  bitstring of length 10, and the GGM tree is a binary tree which branches on each bit of them to lead each of them to a different leaf value.



## Round 4: Message Authentication Code (MAC)

### Oracles and dependencies

1 of 1 ×

Which of the following are true?

(Several answers might be correct. Check all which are correct.)

See link: <http://chrisbrzuska.de/04-quiz-01.pdf>

✓ ☒ a

☐ b

✓ ☒ c

☐ d

☐ e

#### Explanation

(a) is correct since adversary  $A$  makes queries to the EVAL oracle, and (c) is correct because  $R_g$  makes queries to the EVAL oracle. (b) and (e) are wrong, because these sets only contain \*oracle\* names and not package names. (d) is wrong because  $R_g$  does not make a GET query.

### Constructing the reduction

1 of 1 ×

Which of the following 3 reductions  $R_g$  achieves the code-equivalence  $R_g \dashv\dashv Prf_f^0 \dashv\dashv \text{Key}$  is code-equivalent to  $Gprg^0_f$ ?

See this link for the code of  $R_g$ : <http://chrisbrzuska.de/2020-04-quiz.pdf>

☐ a

✗ ☒ b

✓ ☐ c

#### Explanation

(a) does not make use of its oracle. Unlike  $Gprf^0_f$ , it can be called multiple times because it does not return an error. Note that the (a) version of  $R_g$  does not maintain any state and that the variable  $x$  is never assigned.

(b) makes a mistake in the use of the syntax of EVAL. It should only submit a single value to EVAL, the key of EVAL is maintained by  $Prf^0_f$ , not by  $R_g$ .

(c) requests the two PRF values via EVAL and concatenates them, analogously to the construction.

## UNF-CMA

1 of 1 ✕

Would the input-output behaviour of the VERIFY oracle of Unf-cma<sup>1</sup> change if we replace "if  $(x,t) \in L$ " by "if  $(x,t) \in L$  and if  $m.ver(k,x,t)=1$ " ?

✕ ☒ Yes.

✓ ☐ No.

## Explanation

No, because all values  $(x,t)$  in the list have been generated by the MAC oracle, and due to the correctness of the MAC scheme  $m$ , if  $m.mac(k,x)$  returns  $t$ , then  $m.ver(k,x,t)$  returns 1.

## Padding Attack

1 of 1 ✕

Denote by  $c_o$  the last byte of the first ciphertext block. Denote by  $d$  the byte value that needs to be xored onto  $c_o$  to obtain the all-zero byte as the last byte of the second message block. Then, the message byte  $m_o$  of the second message block that we want to recover is equal to...

✓ ☒ ... $d$

☐ ... $d \text{ xor } c_o$

## Explanation

It is equal to  $d$ , because  $d \text{ xor } m_o$  is equal to the all-zero padding. When xoring  $d$  on both sides of the equation  $d \text{ xor } m_o = 0 \dots 0$ , we get that  $d = m_o$ .

## PRFs and MACs

1 of 1 ✕

This construction shows that PRFs and MACs are closely related. What are the differences between a PRF and a MAC? Check all that you think are correct (multiple answers might be correct.). Some of them might ask for your intuition, i.e., we might not have proven these statements yet.

✓ ☒ A PRF output has to be pseudorandom (by definition) while a MAC output "might" be pseudorandom, but the UNF-CMA definition does not demand that the tag looks pseudorandom

✕ ☒ A MAC uses a key while a PRF does not have a key.

✓ ☒ A MAC might append the message to the tag, whereas a PRF cannot reveal even a single bit of its input.

## Explanation

(2) is wrong because a MAC and a PRF both use a key. The PRF part of (3) was proven on Exercise Sheet 3 and the MAC part of (3) will be proven on exercise sheet 4.

## Round 5: Symmetric Key Algorithm (SKE)

### Statements to prove

1 of 1 ✕

The following questions are intended to help you focus on what we seek to prove.  
For all adversaries  $A$ , we try to prove that...

✓ ☐  $\text{Gunf-cma}^0_{\{m_f\}}$  is code-equivalent to  $R_1 \rightarrow \text{Gprf}_f^0$ .

✗ ☒  $\text{Gunf-cma}^1_{\{m_f\}}$  is code-equivalent to  $R_1 \rightarrow \text{Gprf}_f^0$ .

☐  $\text{Gunf-cma}^0_{\{m_f\}}$  is code-equivalent to  $R_2 \rightarrow \text{Gprf}_f^0$ .

✓ ☒  $\text{Gunf-cma}^1_{\{m_f\}}$  is code-equivalent to  $R_2 \rightarrow \text{Gprf}_f^0$ .

☐  $\text{Gunf-cma}^1_{\{m_f\}}$  is code-equivalent to  $R_2 \rightarrow \text{Gprf}_f^1$ .

#### Explanation

Answer 1 is correct and Answer 3 is wrong: We use reduction  $R_1$  to move from the real  $\text{Gunf-cma}^0_{\{m_f\}}$  to a modified game which uses a random function instead.

Answer 2 is wrong: For  $\text{Gunf-cma}^1_{\{m_f\}}$ , we use  $R_2$  and not  $R_1$ .

Answer 4 is correct: We use  $R_2$  to move from an intermediate game which uses a random function instead of a PRF to the ideal  $\text{Gunf-cma}^0_{\{m_f\}}$  game which uses the actual PRF.

Answer 5 is wrong: In fact, neither  $\text{Gunf-cma}^0_{\{m_f\}}$ , nor  $\text{Gunf-cma}^1_{\{m_f\}}$  can be code-equivalent if a truly random function is used by  $\text{Gprf}^1$ .

### Successful distinguisher

1 of 1 ✕

Which of the following distinguishers are successful distinguishers against  $(*, \lambda)$ -PRF-security of the Merkle-Damgard construction? Denote by  $f$  the PRF (also referred to as compression function). Between 0 and 4 answers can be correct.

✓ ☒  $t \leftarrow \text{EVAL}(0^\lambda), t' \leftarrow \text{EVAL}(0^{2\lambda}), \text{ if } t=f(t, 0^\lambda), \text{ return 0, else return 1.}$

✗ ☒  $t \leftarrow \text{EVAL}(0^\lambda), t' \leftarrow \text{EVAL}(1^{2\lambda}), \text{ if } t=f(t, 1^\lambda), \text{ return 0, else return 1.}$

✓ ☒  $t \leftarrow \text{EVAL}(1^\lambda), t' \leftarrow \text{EVAL}(1^{2\lambda}), \text{ if } t=f(t, 1^\lambda), \text{ return 0, else return 1.}$

✓ ☒  $t \leftarrow \text{EVAL}(1^{2\lambda}), t' \leftarrow \text{EVAL}(1^{3\lambda}), \text{ if } t=f(t, 1^\lambda), \text{ return 0, else return 1.}$

#### Explanation

Let us denote by  $\text{MD}_f$  the Merkle-Damgard construction. First notice that in the ideal game  $\text{Gprf}(\text{MD}_f)^1$ , all adversaries return 1 with overwhelming probability. Moreover, all adversary except for the adversary in answer 2 submit values such that in the real game, we would have  $t'=f(t, 0^\lambda)$  (answer 1) or  $t'=f(t, 1^\lambda)$  (answer 3 and 4). However, for answer 2,  $t$  is computed on  $0^\lambda$  while the first iteration of  $t'$  is computed on  $1^\lambda$ , which are unrelated values (since  $f$  is a PRF) and are thus unlikely to be equal.

Check all which are correct (0-4 answers can be correct.).

- ✗ ☒ One-wayness is an adequate way to model the confidentiality of symmetric encryption. (confidentiality: hiding all information about the message)
- ✗ ☒ There are deterministic encryption schemes which are IND-CPA-secure.
- ✗ ☒ A PRP is an IND-CPA-secure encryption scheme for messages of block length  $\lambda$ . (A PRP  $f$  is a pseudorandom function which also has an algorithm  $f_{\text{inv}}$  such that if  $y$  gets  $f(k, x)$ , then  $f_{\text{inv}}(k, y)$  is equal to  $x$ .)
- ✓ ☒ The only state maintained by  $\text{Gind-cpa\_se}^0$  is the key  $k$ .

#### Explanation

Answer 1 is wrong: One-wayness allows to leak even an entire half of the encrypted message. It is hard to argue that this captures confidentiality adequately.

Answer 2 is wrong: Deterministic encryption yields the same ciphertext whenever encrypting the same message twice. As a result, the ENC oracle of the ideal game  $\text{Gind-cpa\_se}^1$  would always return the same ciphertext when encrypting two messages  $m$  and  $m'$  of the same length, while the ENC oracle of the real game  $\text{Gind-cpa\_se}^0$  would return different ciphertexts if  $m$  and  $m'$  are different (this is required for correctness to hold.)

Answer 3 is wrong, since a PRP is deterministic.

## Round 6: Review

OWF

1 of 1 ✕

Check all which are true (can be more than 1). One-way functions...

- ✓ ☒ ... are computable in polynomial time.
- ✓ ☒ ... are deterministic.
- ✓ ☒ ...do not hide very short inputs.
- ✓ ☒ ...might leak a significant part of the input.
- ✗ ☒ ...return a pseudorandom output.

### Explanation

Efficient, deterministic functions cannot hide very short inputs, since given the output, I can perform exhaustive search over the input space and perform an equality check to find a pre-image. If  $f$  is a OWF, then  $f_{\text{new}}(x) := f(x_{\text{left}}) || x_{\text{right}}$  is a OWF which leaks a bit part of its input. If  $f$  is a OWF, then  $f_{\text{zeroes}}(x) := f(x) || 0^{|x|}$  is a OWF, and it does not have pseudorandom output.

PRG

1 of 1 ✕

Check all which apply (can be more than 1). A PRG...

- ✗ ☒ ...is randomized.
- ✓ ☒ ...is computable in polynomial-time.
- ✓ ☒ ...is length-expanding
- ✗ ☒ ...does not leak very short inputs.
- ✓ ☒ ...might leak a big part of the input.
- ✓ ☒ ...returns a bitstring which is indistinguishable from a uniformly random string (provided the input was uniformly random).

### Explanation

PRGs are deterministic (since we use them to gain (pseudo-randomness)). Deterministic functions cannot hide very short inputs, since the code of the PRG is public and thus, given the output, one can perform exhaustive search over a small input space. If  $g$  is a PRG, then  $g_{\text{new}}(x) := g(x_{\text{left}}) || x_{\text{right}}$  is a PRG, too. Moreover,  $g_{\text{new}}$  leaks half of its input.

Check all which are true (can be more than one). PRFs...

- ✓ ☒ ...are deterministic.
- ✓ ☒ ...computable in polynomial-time.
- ✓ ☒ ...are keyed.
- ✓ ☒ ...return the same output, given the same input. (I.e., they allow for equality check).
- ✗ ☒ ...might leak half of their input.

#### Explanation

PRFs cannot leak half of their input, since this would make the output of the PRF distinguishable from a random function, as a random function draws a uniformly random string independently of the input.

Check all which apply (can be more than one). An UNF-CMA-secure MAC...

- ✓ ☒ ...is polynomial-time computable.
- ✗ ☒ ...necessarily has a pseudorandom output.
- ✓ ☒ ...might leak the message.

#### Explanation

If  $m$  is an UNF-CMA-secure MAC scheme, then the following  $m_{\text{new}}$  is also UNF-CMA:  $m_{\text{new}}.\text{mac}(k,x)$  runs  $t \leftarrow m.\text{mac}(k,x)$  and returns  $t_{\text{new}} \leftarrow t \parallel 0$  and  $m_{\text{new}}.\text{ver}(k,x,t_{\text{new}})$  checks that the last bit of  $t_{\text{new}}$  is 0, then removes the last bit of  $t_{\text{new}}$  to get a value  $t$  and runs  $m.\text{ver}(k,x,t)$  and returns its output.  $m_{\text{new}}$  does not have a pseudorandom output.

If  $m$  is an UNF-CMA-secure MAC scheme, we can modify it into a new UNF-CMA-secure MAC scheme which appends the message to the tag (and thus leaks the message).

Check all which apply (can be more than one).

- ☒ ☐ A symmetric encryption scheme can be computed in deterministic polynomial-time.
- ☒ ☐ A symmetric encryption scheme can be computed in probabilistic polynomial-time.
- ☒ ☐ When running an IND-CPA secure symmetric encryption scheme twice on the same key and the same message, I will always get the same output.
- ☒ ☐ An IND-CPA symmetric encryption scheme hides the content (but not the length) of short messages.
- ☒ ☐ An AE-secure symmetric encryption scheme hides the content (but not the length) of short messages.
- ☒ ☐ In addition to confidentiality, AE also provides integrity/authenticity guarantees.

#### Explanation

A symmetric encryption scheme cannot be computed by a deterministic algorithm since it is a randomized algorithm. Since it is randomized, it will yield different outputs (with high probability), even when run on the same inputs. The security notions of IND-CPA and AE both capture that the encryption of the real message is indistinguishable from the encryption of a all-zeroes message of the same length and thus, IND-CPA and AE-secure encryption schemes also hide the content of short messages (but not their length).

## Round 7: Public Key Encryption and Signature Schemes (PKE & SIG)

### IND-CPA definition

1 of 1 ✕

Check all which apply.

- ✓ ☒ One difference between the IND-CPA definition for symmetric-key encryption and public-key encryption is that in the IND-CPA security game for public-key encryption, the adversary additionally gets the public-key.
- ✓ ☒ In public-key encryption, the adversary can easily generate valid ciphertexts on its own.
- ✗ ☒ The encryption algorithm of a public-key encryption scheme is deterministic.
- ✗ ☒ The GETPK oracle returns the secret key  $sk$  to the adversary.

#### Explanation

Answer 1: This is indeed the main difference and captures how symmetric-key encryption and public-key encryption differs conceptually.

Answer 2: Since the adversary knows the public-key (it can obtain it via the GETPK oracle), it can use the public-key to run the public-key encryption algorithm on a message of its choice.

Answer 3: Analogously to the symmetric-key encryption case, it is important that the encryption algorithm is randomized since else, the ciphertexts would leak when a message repeats. (See exercise sheet this week: There is an easier attack in the PKE case which only makes a single encryption query whereas in the SE case, we needed to make two encryption queries in the attack.)

Answer 4: It is very important that the secret key  $sk$  remains hidden from the adversary. If the adversary knew the secret-key, it could decrypt the ciphertext and easily distinguish the real and the ideal game, e.g., by querying  $c \leftarrow \text{ENC}(1^\lambda)$  and then decrypting  $c$ . If the decryption yields  $1^\lambda$ , it is the real game. If the decryption yields  $0^\lambda$ , it is the ideal game.

### IND-CPA-secure PKE $\Rightarrow$ OWF

1 of 1 ✕

The argument here implicitly relies on the fact that if  $\text{pke.enc}(pk, x, r) = c$  and  $x$  is not equal to  $0 \dots 0$ , then there is no other randomness  $r'$  such that  $\text{pke.enc}(pk, 0 \dots 0, r') = c$ . Why is this the case? (Check all which are correct.)

- ✓ ☒ Consider a fixed  $pk$ . If  $x$  is different from  $0 \dots 0$ , then the ciphertexts of  $x$  and the ciphertexts of  $0 \dots 0$  are disjoint sets.
- ✗ ☒ Encryption is deterministic.
- ✗ ☒ If  $\text{pke.enc}(pk, x, r) = c$ , then decryption  $\text{pke.dec}(sk, c)$  returns  $x$  by correctness of the public-key encryption scheme. If there is some  $r'$  such that  $\text{pke.enc}(pk, 0 \dots 0, r') = c$ , then by the correctness of the decryption scheme,  $\text{pke.dec}(sk, c)$  returns  $0 \dots 0$ . But  $x$  is not equal to  $0 \dots 0$  and  $\text{pke.dec}(sk, c)$  can thus only return  $x$  or  $0 \dots 0$  but not both at the same time.

#### Explanation

Answer 3 is pretty much the explanation already. In particular, answer 3 implies answer 1. Namely, we cannot have a ciphertext which can both be an encryption of  $x$  and an encryption of  $0 \dots 0$  because this would violate the correctness criterion. (Note that some people define correctness to only hold with overwhelming probability, and then, the ciphertext sets of different messages might overlap.)

Answer 2 is unrelated, but it's also wrong, since the encryption algorithm needs to be probabilistic as else, the encryption scheme would be insecure (see this week's exercise sheet).



Check all which apply.

- ✓ ☒ Existence of IND-CPA secure public-key encryption implies existence of one-way functions in a black-box way.
- ✗ ☒ Existence of one-way functions implies existence of IND-CPA secure public-key encryption in a black-box way.

#### Explanation

Pihla showed us a proof that  $\text{PKE} \Rightarrow \text{OWF}$  which is a black-box proof and thus answer 1 is correct. However, Pihla showed us a claim that that OWFs do not imply PKEs in a black-box way and thus, answer 2 is false.

Check all which apply.  $(pk, sk)$  denotes the public-key, secret-key pair of the hybrid public-key encryption scheme, and  $k$  denotes the symmetric-key which is sampled in the encryption algorithm.

- ✗ ☒ In hybrid public-key encryption, the secret-key  $sk$  is encrypted under the symmetric-key  $k$ .
- ✓ ☒ In hybrid public-key encryption, the symmetric key  $k$  is encrypted under the public-key  $pk$ .
- ✗ ☒ In hybrid public-key encryption, the encryption algorithm uses the secret key  $sk$ .
- ✗ ☒ Encryption uses the same symmetric-key  $k$ .

#### Explanation

Answer 1 and 3 are wrong: Encryption is supposed to be a public operation and thus, it cannot depend on the secret-key.

Answer 2 is correct: The encryption algorithm draws a symmetric-key  $k$  which is then encrypts under the public-key  $pk$ .

Answer 4 is wrong: If the symmetric-key was the same, then it would be public information and hence, the scheme would be insecure.

Check all which apply.

- ✓ ☒ The sum of three negligible functions is negligible.
- ✗ ☒ If an adversary can distinguish  $G_{\text{ind-cpa}}^0$  and  $G_{\text{ind-cpa}}^1$  with non-negl. advantage, then the distinguisher can distinguish with non-negligible advantage between  $\text{Hyb}^1$  and  $\text{Hyb}^2$ .
- ✓ ☒ If an adversary can distinguish  $G_{\text{ind-cpa}}^0$  and  $G_{\text{ind-cpa}}^1$  with non-negl. advantage, then the distinguisher can distinguish with non-negligible advantage "at least" one of the three game-hops.
- ✓ ☒ In this game-hopping proof, we used the IND-CPA security of the PKE twice.
- ✗ ☒ In this game-hopping proof, we used the IND-CPA security of the symmetric-encryption scheme twice.

#### Explanation

Answer 1: Correct. (This was an exercise on an exercise sheet before.)

Answer 2: Not necessarily correct, because we do not know against which of the game-hops the adversary is successful, only that he is successful against one of them.

Answer 3: True. Assume towards contradiction that Answer 3 is false, then we can bound the advantage of the adversary against  $G_{\text{ind-cpa}}^0$  and  $G_{\text{ind-cpa}}^1$  by the sum of the 3 advantages against the 3 game-hops. Since these advantages are all negligible, and the sum of 3 negligible functions is negligible (cf. Answer 1), we bounded a non-negligible function by a negligible function. Contradiction.

Answer 4 and 5: The first and last game hop reduce to IND-CPA security of the PKE and the middle game hop reduces to the IND-CPA security of the symmetric-key encryption scheme.

Check all which apply.

- ✗ ☒ The MAC oracle returns the key  $k$ .
- ✓ ☒ Unforgeability under chosen message attacks models that no adversary can come up with "new" message-tag pairs (i.e., those which did not come from the MAC oracle) which pass verification.
- ✓ ☒ Unforgeability is modeled by having a list in the ideal game so that VERIFY really only returns 1 when a message-tag pair came from a query to the MAC oracle.

#### Explanation

Answer 1: Wrong. The MAC oracle returns the tag  $t$ . It is important for security that the key  $k$  remains secret.

Answer 2/3: Correct. The answers are essentially explanations of the UNF-CMA game...

## Signature scheme syntax

1 of 1 ✕

Check all which apply.

- ☒ ✕ Signatures are verified using the secret-key sk.
- ☒ ✓ Signatures are verified using the public-key pk.
- ☒ ✓ Signatures over messages are generated using the secret-key sk.
- ☒ ✕ Signatures over messages are generated using the public-key pk.

### Explanation

Answer 1/2: The s.verify algorithm models signature verification and takes as input the public-key.

Answer 3/4: The s.sig algorithm models signature generation and takes as input the secret-key.

The idea here is, roughly, an analogue to handwritten signatures: Only the signer can sign (using the secret key or having his own way of generating a signature), but everyone can verify a signature (using the public-key or using an ID card). The analogy with handwritten signatures is not too good, because cryptographic signatures are much better: For example, when I have a handwritten signature on a document, someone can still modify the text before the signature. With a cryptographic signature, such a change would be detected.

## Lamport's signature scheme

1 of 1 ✕

Check all which apply.

- ☒ ✓ The existence of one-way functions implies the existence of UNF-CMA-secure signature schemes in a black-box way.
- ☒ ✕ The existence of one-way functions does not imply the existence of UNF-CMA-secure signature schemes in a black-box way.
- ☒ ✓ In Lamport's signature scheme, the secret-key consists of  $2\lambda$  bitstrings of length  $\lambda$ .
- ☒ ✓ In Lamport's signature scheme, a signature over a message of length  $\lambda$  consists of  $\lambda$  bitstrings of length  $\lambda$ .
- ☒ ✓ Given one message-signature pair, it is hard to find a signature for a different message, because a different message differs from the signed message on at least one bit, and the adversary does not know the pre-image of the  $y$  for that bit.

### Explanation

Answer 1/2: Lamport's signature scheme is a way of transforming any (injective) OWF into a signature scheme.

Answer 3-5 are all correct and essentially describe properties of Lamport's signature scheme.

## Round 8: Lattices

### Average-case problems

1 of 1 ✕

Which of the following is/are average case problem/s?

✓ ☒ Breaking the one-wayness of one-way functions

✗ ☒ SVP

✓ ☒ SIS

✓ ☒ LWE

✗ ☒ CVP

✗ ☒ Circuit satisfiability

#### Explanation

Breaking one-wayness is an average case problem because the adversary is given  $\text{OWF}(x)$  for a random  $x$ .

### Norm

1 of 1 ✕

Let  $x$  be a random vector drawn uniformly at random from  $\mathbb{Z}_q^n$ . What can we say about the infinity norm of  $x$ ?

✗ ☒  $\|x\| > 0$

✓ ☒  $\|x\| \geq 0$

✗ ☒  $x = (q-1, q-1, \dots, q-1)$  is a possible outcome and  $\|x\| = q-1$

✓ ☒  $\|x\| \leq q/2$

#### Explanation

With the balanced representation, any vector  $x$  in  $\mathbb{Z}_q^n$  will have norm between 0 and  $\lfloor q/2 \rfloor$ .

## SIS problem

1 of 1 ✕

Which of the following statements is/are true?

- ☒ ✕ For any choice of parameters, a SIS problem always has a solution with non-negligible probability.
- ☒ ✓ There exists a choice of parameters such that the SIS assumption is false.
- ☒ ✓ If  $m > n \log_{\beta} q$ , then the  $\text{SIS}_{\{n,m,q,\beta,0\}}$  problem has at least one solution.
- ☒ ✕ The  $\text{SIS}_{\{n,m,q,\beta,0\}}$  problem is trivial because  $u = 0$  is always a solution.

### Explanation

1) If  $m < n$ , the SIS problem likely does not have a solution since the system of equations is likely over-determined. 2) If  $m > n \log_{\beta} q$ , then  $A$  is likely full-rank. Let  $v = 0$ . If  $A$  is full-rank, there must exist a non-zero  $u$  in  $\mathbb{Z}_q^m$  such that  $Au = 0 \pmod q$ . If  $\beta = q/2$ , then any vector  $u$  in  $\mathbb{Z}_q^m$  satisfies  $\|u\| \leq \beta$  trivially. 3) Since  $m > n \log_{\beta} q$ , there are more vectors in  $\mathbb{Z}_{\beta}^m$  than in  $\mathbb{Z}_q^n$ . By the pigeonhole principle, there exists distinct  $u, u'$  in  $\mathbb{Z}_{\beta}^m$  such that  $Au = Au' \pmod q$ . Furthermore the norm of  $u - u'$  is at most  $\beta$ . Thus  $u - u'$  is a solution. 4) The statement is false because  $0 < \|u\|$  rules out this trivial solution.

## SIS v.s. LWE

1 of 1 ✕

Which of the following is a (more) correct statement?

- ☐ The LWE problem is at least as hard as the SIS problem because if one could find a short vector  $u$  such that  $Au = 0 \pmod q$  then one could solve LWE by computing  $b^T u \pmod q$  and checking whether it is small.
- ☒ ✓ The SIS problem is at least as hard as the LWE problem because if one could find a short vector  $u$  such that  $Au = 0 \pmod q$  then one could solve LWE by computing  $b^T u \pmod q$  and checking whether it is small.

### Explanation

"Problem  $P$  is as hard as problem  $Q$ " is just a fancy way of saying that there exists a reduction from  $Q$  to  $P$ . The second part of the statement, i.e. because ..., gives such a reduction. The reduction works because if  $b^T = s^T A + e^T \pmod q$  then  $b^T u = (s^T A + e^T) u = s^T Au + e^T u \pmod q$ , which is short. Else, if  $b$  is random then  $b^T u$  is likely not short. In other words, an SIS oracle allows us to solve decision-LWE. From the lecture, we know that decision-LWE is at least as hard as search-LWE.

## Regularity Lemma

1 of 1 ✕

The regularity property holds for  $(n,m,q,\beta)$  only if the  $\text{SIS}_{\{n,m,q,\beta\}}$  assumption holds.

- ☒ ✕ True
- ☐ ✓ False

### Explanation

The statement is false because the regularity property holds as long as  $m > n \log_{\beta} q$ . It is not conditioned on any computational assumption.

## Correctness of Regev's encryption

1 of 1 ✕

Which of the following modifications allow(s) encrypting more than a single bit, i.e. how to encrypt  $x = (x_0, \dots, x_{k-1})$  where each  $x_i$  is a bit?

- ✓ ☒ Encrypt each  $x_i$  independently and concatenate the results.
- ✓ ☒ Change the key generation algorithm so that  $B = SA + E \bmod q$ , where  $B$ ,  $S$ , and  $E$  each now consists of  $k$  rows. Change the encryption algorithm so that  $c_1$  is computed as  $c_1 = Br + \text{floor}\{q/2\} x \bmod q$ . Note that  $c_1$  and  $x$  are now vectors.
- ✗ ☒ Pack the bits  $x_0, \dots, x_{k-1}$  into a single integer  $x' = x_0 + 2x_1 + \dots + 2^{k-1}x_{k-1}$ . Change the encryption algorithm so that  $c_1$  is computed as  $c_1 = b^T r + \text{floor}\{q/2\} x' \bmod q$ .

### Explanation

1) and 2) are easy to verify. 3) leads to unpredictable decryption results because  $\text{floor}\{q/2\} x'$  might be greater than  $\text{floor}\{q/2\}$  which triggers modular reduction.

## Security of Regev's Encryption

1 of 1 ✕

In the security proof of Regev's encryption, we could alternatively first swap the ciphertext to a uniformly random one using the regularity lemma, then swap the public key to a uniformly random one using the LWE assumption.

- ✗ ☒ True
- ✓ ☐ False

### Explanation

Without first swapping the public key, the vector  $b$  is structured, i.e.  $b^T = s^T A + e^T \bmod q$  for some  $s$  and short  $e$ . Since this property does not generally hold for randomly sampled  $b$ , the regularity lemma cannot be applied.

## Round 9: Fully Homomorphic Encryption (FHE)

### Intuitive security of FHE

1 of 1 ✕

Although Bob does not learn  $x$ , it learns the computation result  $f(x)$ .

✕ ☒ True

✓ ☐ False

#### Explanation

Bob should not learn anything about  $x$ , including the value of  $f(x)$  which depends on  $x$ . More formally, as we will discuss later, an FHE should also satisfy IND-CPA-security. This means Bob should not be able to distinguish between  $ctxt_x$  and  $ctxt_y$  for any  $y$ . Therefore anything that Bob could learn from  $ctxt_x$  should also be learnable from  $ctxt_y$  for any  $y$ .

### PKE with Bilinear Decryption

1 of 1 ✕

A PKE with bilinear decryption cannot be secure because the secret key can be recovered by

✕ ☒ brute force.

✓ ☐ solving a system of linear equations.

☐ frequency analysis.

☐ side-channel attacks.

#### Explanation

By encrypting several known random messages and considering the decryption function restricted at these ciphertexts as a function of the secret key, we obtain a system of linear equations whose solution is the secret key. For more details, please refer to the lecture notes.

### FHE from PKE with Bilinear Decryption

1 of 1 ✕

Apart from the attack in the previous quiz, the FHE scheme  $\Pi'$  is trivially broken in yet another way because

✕ ☒ a ciphertext of the secret key is given out as part of the public key.

✓ ☐ the encryption algorithm is deterministic.

#### Explanation

A deterministic encryption scheme, fully homomorphic or not, cannot be IND-CPA-secure.

## Noise Growth in FHE

1 of 1 ✕

What is a rough upper bound of the error of  $\text{ctxt}_{xy}$  if  $\text{ctxt}_x$  has error  $e_x$  and  $\text{ctxt}_y$  has error  $e_y$ ?

- ✗ ☒  $\max\{e_x, e_y\} * n * \log q$
- ☐  $e_x * n * \log q + x * e_y$
- ✓ ☐  $e_y * n * \log q + y * e_x$
- ☐  $(e_x + e_y) * n * \log q / 2$

### Explanation

The " $e_y * n * \log q$ " term comes from summing up " $a_{h,k} * \text{ctxt}_y * sk_h * 2^k$ " for  $h$  from 1 to  $n$  and for  $k$  from 0 to  $\ell \approx \log q$ . The " $y * e_x$ " term comes from " $y * L_{ij}(sk) = xy + y * e_x$ ". Note that when  $y$  is small (e.g. binary) and  $e_x \approx e_y$  then the first term dominates the second.

## Gadget Matrix

1 of 1 ✕

Which of the following is/are true?

- ✗ ☒ For any matrix  $X$  of appropriate dimensions, we have  $G^{-1}(X) * G = X \bmod q$ .
- ✗ ☒ Let  $n = 1$  and  $q = 11$  so that  $G = \begin{pmatrix} 1 & 2 & 4 & 8 \end{pmatrix}$ . We have  $G^{-1}(5) = \begin{pmatrix} 0 & 1 & 0 & 1 \end{pmatrix}^T$ .
- ✓ ☒ Let  $n = 1$  and  $q = 11$  so that  $G = \begin{pmatrix} 1 & 2 & 4 & 8 \end{pmatrix}$ . We have  $G^{-1}(5) = \begin{pmatrix} 1 & 0 & 1 & 0 \end{pmatrix}^T$ .
- ✗ ☒  $G^{-1}(\cdot)$  is a linear operation since it is the inverse operation of the left-multiplication of  $G$ , which is a linear operation.

### Explanation

For Answers 1 and 4, it is easy to come up with counterexamples. For Answers 2 and 3, mind the ordering of the powers of 2 in  $G$ .

## Homomorphic Addition and Multiplication

1 of 1 ✕

Why do the homomorphic evaluations only almost work but not really?

- ✓ ☒  $1+1=2$ .
- ☐ After homomorphic multiplication, the norm of the encryption randomness grows beyond  $q/4$ .
- ☐ The matrix dimensions don't match.

### Explanation

Continue watching to see why.



## Round 10: Multi-party Computation (MPC)

### Possible Constructions

1 of 1 ✕

Check all which you believe are true. (We here use the term "build" as a short way to refers to the term "construct in a black-box way".)

- ✓ ☒ We can build a secure PRG from a secure OWF.
- ✗ ☒ We can build an IND-CPA secure PKE from a secure OWF
- ✓ ☒ We can build an IND-CPA secure symmetric encryption scheme from a secure OWF.
- ✗ ☒ We can build an secure FHE scheme from a secure OWF.

#### Explanation

- 1: Yes, all primitives in the green box can be turned into one another. For this specific implication, see Lecture 2.
- 2: No. From IND-CPA-secure PKEs, we can construct one-way functions, but from one-way functions, we cannot construct PKE. In Lecture 7, we saw that there is a separation.
- 3: Yes, all primitives in the green box can be turned into one another. For this specific implication, see Lecture 2-5.
- 4: No. From secure FHE, we can build one-way functions, but from one-way functions, we cannot construct PKE. See lecture 7 and 8.

### Double-and-add algorithm

1 of 1 ✕

The double-and-add algorithm multiplies a key (which is an integer represented by a bitstring)  $k$  with a point  $P$  by performing operations for each bit of  $k_i$ . Check all which you believe are true.

- ✗ ☒ When the  $i$ -th bit  $k_i$  is 0, double-and-add performs two additions.
- ✗ ☒ When the  $i$ -th bit  $k_i$  is 0, double-and-add performs a doubling operation and an addition.
- ✗ ☒ When the  $i$ -th bit  $k_i$  is 0, double-and-add performs only an addition.
- ✓ ☒ When the  $i$ -th bit  $k_i$  is 0, double-and-add performs only a doubling.
- ✗ ☒ When the  $i$ -th bit  $k_i$  is 1, double-and-add performs two additions.
- ✓ ☒ When the  $i$ -th bit  $k_i$  is 1, double-and-add performs a doubling operation and an addition.
- ✗ ☒ When the  $i$ -th bit  $k_i$  is 1, double-and-add performs only an addition.
- ✗ ☒ When the  $i$ -th bit  $k_i$  is 1, double-and-add performs two additions.

#### Explanation

Double-and-add always performs a point doubling (regardless of what the bit  $k_i$  is). Thus, Answer 3 and 7 are incorrect. If the bit  $k_i$  is 0, then no other operations are performed, and thus, Answer 4 is correct (and Answer 1 and 2 are wrong). If the bit  $k_i$  is 1, then additionally to the point doubling, an addition is performed, and thus, Answer 6 is correct (and Answer 5 and 8 are wrong).

This trace corresponds to first computing a 0, then a 1, then...

- ☐ ...another 1, then another 0 and then another 0.
- ✕ ☒ ...another 0, then another 1 and then another 1.
- ✓ ☐ ...another 0, and another 1.
- ☐ ...another 1, and another 0
- ☐ ...another 1 and another 1.
- ☐ ...another 0 and another 0.

#### Explanation

0 = only short interval, 1 = short interval followed by long interval, thus the trace corresponds to 0101, since it starts with two short intervals and 1 long interval (corresponding to 0 and then 1) and then again two short intervals and 1 long interval (corresponding to 0 and then 1).

Check all which you believe are true:

- ✕ ☒ When the  $i$ -th bit  $k_i$  is 0, double-and-add-always performs a two additions.
- ✓ ☒ When the  $i$ -th bit  $k_i$  is 1, double-and-add-always performs one doubling operation and one addition.
- ✕ ☒ When the  $i$ -th bit  $k_i$  is 0, double-and-add-always only performs a point doubling and no addition.
- ✓ ☒ When the  $i$ -th bit  $k_i$  is 0, double-and-add-always performs one doubling operation and one addition.
- ✓ ☒ double-and-add-always performs one doubling operation and one addition regardless whether the  $i$ -th bit  $k_i$  is 0 or 1.

#### Explanation

Answer 2, 4 and 5 are true. The main idea of double-and-add-always is indeed that the algorithms always performs one point doubling and one addition, regardless of the bit value  $k_i$ . Note, however, that sometimes, the result of the add operation is ignored --- yet, it is still computed which is relevant to prevent the attack we saw before.

## Montgomery Ladder

1 of 1 ✕

Check all which apply.

- ✓ ☒ Regardless of the bit  $k_i$ , the Montgomery ladder always performs one addition and one point-doubling.
- ✗ ☒ For bit  $k_i = 0$ , the Montgomery ladder performs only a point doubling.
- ✓ ☒ In the Montgomery Ladder, the point doubling and the point addition can be implemented in parallel since they have no dependencies.

### Explanation

The idea of the Montgomery ladder is to perform the same sequence of operations regardless of whether a 0 or 1 is processed. Answer 1 is correct. Answer 2 is incorrect, since the else branch contains an add instruction.

Answer 3: Indeed, the main advantage of the Montgomery ladder over the double-and-add-always algorithm is that the point doubling and the add operation can be performed in parallel.

## Candidates

1 of 1 ✕

From which time interval can we infer a key out of which 80% or more of the bits are correct? Check all which you believe are correct.

- ✓ ☒ 1
- ✗ ☒ 2
- ✗ ☒ 7
- ✗ ☒ 10
- ✗ ☒ 15
- ✗ ☒ 32
- ✓ ☒ 47
- ✓ ☒ 48
- ✗ ☒ 53
- ✓ ☒ 56
- ✓ ☒ 57

### Explanation

The slots 1, 47, 48, 56 and 57 are all helpful to obtain a 80% correct key. The most interesting part, here, is slot 56, since the key values need to be flipped. That is, an attack that is wrong on 80% of the bits is actually a useful attack :-)

The use of randomized encodings does not change the input-output behaviour of the white-box program.

✓ ☒ True

☐ False

**Explanation**

This is true, since, e.g., the encoding  $f_1$  and  $f_1^{-1}$  cancel out. In general, it is very important that any white-box techniques that we use preserve the behaviour of the white-box program.

## Round 11: Secure Multi-party Computation (MPC)

### Secret Sharing Security Definition

1 of 1 ✕

Check all which you are believe to be true.

- ☒ ☐ The simulator gets as input the secret  $s$ .
- ☒ ☐ The simulator needs to produce  $t$  shares which are then given to the adversary.
- ☒ ☐ The simulator needs to produce  $t-1$  shares which are then given to the adversary..
- ☒ ☐ The secret-sharing security definition models confidentiality of the secret  $s$ .

#### Explanation

Answer 1 is incorrect. Since we want to model confidentiality of the secret (Answer 4 is correct), in the ideal world, what we return to the adversary does not depend on the secret  $s$ .

Answer 2 is incorrect, since with  $t$  shares, the adversary could reconstruct the secret  $s$ , and the simulator does not know the secret  $s$ . Hence, the simulator only returns  $t-1$  shares, as Answer 3 states correctly.

### Secret-Sharing Simulator

1 of 1 ✕

Check all which you believe are true. (Some questions can be considered advanced questions.)

- ☒ ☐ The simulator returns  $t-1$  uniformly random bitstrings.
- ☒ ☐ The simulator returns  $t$  uniformly random bitstrings.
- ☒ ☐ In the real encoding, each subset of  $t-1$  of the  $sh_1, \dots, sh_t$  are distributed as  $t-1$  uniformly random, independent strings, i.e., strings which are independent from each other and from the secret  $s$ .
- ☒ ☐ In the real encoding, the  $sh_1, \dots, sh_t$  are distributed as  $t$  uniformly random, independent strings, i.e., strings which are independent from each other and from the secret  $s$ .

#### Explanation

Answer 1 is correct (and Answer 2 is false): With  $t$  bitstrings, the adversary could already recover the secret  $s$  (which the simulator does not have), thus the simulator only simulates  $t-1$  strings. Answer 4 is incorrect since  $sh_t$  can be computed given  $s$  and  $sh_1, \dots, sh_{t-1}$ . So, it is not independently uniformly distributed (advanced). Answer 3 is indeed true, we will see the argument in the lecture video shortly (advanced).

## Protocol for Computing AND of three bits

1 of 1 ✕

Check all which you believe are true.

- ✓ ☒ In this protocol, Penguin A sends two messages.
- ✓ ☒ In this protocol, Penguin B sends two messages.
- ✓ ☒ If all penguins have 1 as their own bit, then  $r_0^A = r_1^A$ ,  $r_0^B = r_1^B$ ,  $r_0^C = r_1^C$ .
- ✓ ☒ If all penguins have 1 as their own bit, then  $r \text{ xor } r_1^C$  is the all-zeroes string.
- ✓ ☒ Penguin A does not learn anything in this protocol except for the result of the computation.

### Explanation

Everyone receives two messages, except for penguin A who only receives one since penguin A starts the protocol. Everyone sends two messages except for penguin C who only sends one since penguin C ends the protocol. Thus, Answer 1 is correct and Answer 2 is false.

Answer 4 is correct, since  $r \text{ xor } r_1^C$  is the xor of all the strings listed in Answer 3 (which is correct) and thus, they all cancel out.

Answer 5 is correct (see how the discussion continues). The reason is that the message from C is a uniformly random bitstring.

## What does penguin C learn?

1 of 1 ✕

Check all which you believe are true.

- ✓ ☒ If penguin C has bit 0, penguin C learns whether penguin A and B both had a 1-bit.
- ✓ ☒ If penguin C has bit 1, penguin C learns whether penguin A and B both had a 1-bit.
- ✗ ☒ If penguin C has bit 0, then from the AND of the three bits of all penguins, penguin C knows whether penguin A and B both had a 1-bit.
- ✓ ☒ If penguin C has bit 1, then from the AND of the three bits of all penguins, penguin C knows whether penguin A and B both had a 1-bit.

### Explanation

(1) and (2) are both true, because \*regardless of the bit of penguin C\*, penguin C learns whether penguin A and B both had a 1-bit. If this information was already leaked from the result of the AND computation, then this would be fine. Namely, if all penguins have bit 1, then the output of the protocol is 1, and then, they immediately know that all other penguins had bit 1. However, if the result is 0, then they don't know which bits the other penguins had. In particular, if a penguin's own bit is 0, then it does not learn anything about the bits of the other penguins, since the result is 0, regardless of what the secret bit of the other penguins is (that's why (3) is wrong.).

When running a 1-out-of-4-OT protocol, which information does Alice learn?

- ✗ ☒ Alice learns which bit Bianca learned.
- ✓ ☐ Alice does not obtain any information from running the OT protocol.

#### Explanation

Indeed, one of the reasons that this transfer protocol is called oblivious is that the sender does not learn which of the bits it sent were transferred to the receiver.

Which information does Bianca learn from running the 1-out-of-4-OT protocol. Check all which you believe are true.

- ✗ ☒ Bianca learns 2 bits, namely  $s_{x0}$  and  $s_{x1}$
- ✓ ☒ Bianca learns 1 bit, namely  $s_{x0x1}$ .
- ✓ ☒ If Bianca's secret bits are 00, then Bianca learns  $s_{00}$ .
- ✓ ☒ If Bianca's secret bits are 00, then Bianca does not learn  $s_{01}$ .

#### Explanation

The functionality of the 1-out-of-4-OT protocol is that it should transfer one out of four strings to Bianca, namely the one that corresponds to the secret bits of Bianca. Thus, answer 2 and 3 are correct. Moreover, Bianca does not learn anything about the other bits and thus, 4 is correct, too.

## Round 12: Summary

### Recap

1 of 1 ✕

Check all which you believe are correct.

- ☒ ☐ A secure one-way function necessarily returns a random-looking output.
- ☒ ☐ For a secure one-way function, it is hard to find a pre-image when given the output of the one-way function on a uniformly random input.
- ☒ ☐ A secure pseudorandom generator necessarily returns a random-looking output when receiving a uniformly random input.
- ☒ ☐ There exist secure pseudorandom generators which leak half of their input.

#### Explanation

Answer 1 is incorrect. Recall that if  $f$  is a secure one-way function, then also  $g(x) := f(x) || 0..0$  is a secure one-way function, although the output is not random-looking. Answer 2-4 are correct. 2 and 3 are essentially the definitions of OWFs and PRGs. Answer 4 is true as well, since if  $g$  is a secure PRG, then  $h(x_{\text{left}} || x_{\text{right}}) := g(x_{\text{left}}) || x_{\text{right}}$  is a secure PRG as well.

### PRFs

1 of 1 ✕

Check all which you believe to be true.

- ☒ ☐ A pseudorandom function deterministically maps a key  $k$  and an input  $x$  to an output  $y$ .
- ☒ ☐ A pseudorandom function is a randomized algorithm. In particular, when running  $\text{prf}(k, x)$  twice, with high probability, I obtain different outputs.

#### Explanation

Answer 2 is incorrect. In particular, such a cipher would be extremely inconvenient. See, e.g., countermode: The decryptor would obtain a different message than the encryptor.

### Signature Schemes

1 of 1 ✕

Check all which you believe are true.

- ☒ ☐ All UNF-CMA secure signature schemes provide integrity protection.
- ☒ ☐ All UNF-CMA secure signature schemes provide confidentiality of the message that was signed.
- ☒ ☐ We can build an UNF-CMA secure signature scheme from a one-way function (and a collision-resistant hash-function).
- ☒ ☐ We can build a secure one-way function from an UNF-CMA secure signature scheme.

#### Explanation

Answer 1 is true by definition of UNF-CMA security. Answer 2 is not correct, see video for explanation. Answer 3 and 4 are true, see video for explanation.



## Public-Key Encryption

1 of 1 ✕

Select all which you believe to be true.

- ✓ ☒ An IND-CPA secure encryption scheme necessarily provides confidentiality.
- ✗ ☒ An IND-CPA secure encryption scheme necessarily provides unforgeability, i.e., it is hard for an adversary to create a valid ciphertext for a message of its choice.
- ✗ ☒ It is known how to build an IND-CPA-secure public-key encryption scheme from any secure one-way function.
- ✓ ☒ It is known how to build a one-way function from any IND-CPA-secure public-key encryption scheme.

### Explanation

Answer 1 is correct by definition of IND-CPA (for an explanation, rewind the video by 5 minutes). Answer 2 and 3 are false, and answer 4 is correct. Explanations will be given in the video.

## Secure Multi-Party Computation

1 of 1 ✕

Check all which you believe are correct.

- ✓ ☒ Secure multi-party computation allows several parties with secret inputs  $x_1, \dots, x_n$  to evaluate a function  $f$  jointly such that everyone learns the outcome  $y=f(x_1, \dots, x_n)$ , but nothing else about the secret inputs  $x_1, \dots, x_n$ .
- ✓ ☒ A 1-out-of-2 oblivious transfer protocol allows Alice to transmit one out of 2 strings to Bianca such that (a) Alice does not know "which" of the two strings Bianca learned and (b) Bianca does not learn any information about the second string.
- ✗ ☒ We know how to build an oblivious transfer protocol from one-way functions.
- ✓ ☒ We know how to build a one-way function from any secure oblivious transfer protocol.

### Explanation

Answer 1 and 2 are essentially the (informal) definitions of secure multi-party computation and 1-out-of-2 oblivious transfer.

Answer 3 is wrong. Oblivious transfer protocols imply secure key agreement protocols which are very similar to public-key encryption and places them in "CryptoMania". We have a separation result (Pihla showed this to us) establishing that it is very hard to build primitives in CryptoMania from one-way functions.

Answer 4 is correct, since any computationally secure cryptographic primitive implies one-way functions.