

# **Exercises**

## **Cryptography**

# Question 1

You are having a discussion with a friend about cryptography. Your friend makes a series of statements. Please tell us how you would respond (True or False) and motivate your answer.

- A. No encryption algorithm is perfect, as they are all vulnerable to brute forcing.
- B. An encryption algorithm is broken when there is at least one way to derive the key from a given amount of ciphertext.
- C. The reason why the 2048bit RSA is more robust to brute forcing than a 256bit AES, is because the key is longer.

A. No encryption algorithm is perfect, as they are all vulnerable to brute forcing.

*[see slides on “cryptography”] False. A perfect cipher, i.e., a one-time pad with a key chosen uniformly at random is not vulnerable to an exhaustive search as all the keys will yield a potentially meaningful plaintext (with the same likelihood of being the good one).*

B. An encryption algorithm is broken when there is at least one way to derive the key from a given amount of ciphertext.

A. No encryption algorithm is perfect, as they are all vulnerable to brute forcing.

*[see slides on “cryptography”] False. A perfect cipher, i.e., a one-time pad with a key chosen uniformly at random is not vulnerable to an exhaustive search as all the keys will yield a potentially meaningful plaintext (with the same likelihood of being the good one).*

B. An encryption algorithm is broken when there is at least one way to derive the key from a given amount of ciphertext.

*[see slides on “cryptography”] False. Generally, a cryptosystem is broken if there is any attack faster than trying all the possible keys to obtain the plaintext.*

C. The reason why the 2048bit RSA is more robust to brute forcing than a 256bit AES, is because the key is longer.

A. No encryption algorithm is perfect, as they are all vulnerable to brute forcing.

*[see slides on “cryptography”] False. A perfect cipher, i.e., a one-time pad with a key chosen uniformly at random is not vulnerable to an exhaustive search as all the keys will yield a potentially meaningful plaintext (with the same likelihood of being the good one).*

B. An encryption algorithm is broken when there is at least one way to derive the key from a given amount of ciphertext.

*[see slides on “cryptography”] False. Generally, a cryptosystem is broken if there is any attack faster than trying all the possible keys to obtain the plaintext.*

C. The reason why the 2048bit RSA is more robust to brute forcing than a 256bit AES, is because the key is longer.

*[see slides on “cryptography”] False. The size of the key cannot be used as a direct comparison criterion because the required computational effort depends on the key length in a different way depending on the cipher.*

## Question 2


Consider a data protection mechanism which encrypts an entire hard disk, block by block, by means of AES in Counter (CTR) mode, employing a 128 bit key.

The system administrator, following a new directive which mandates keys to be at least 256 bits long, implements the following compatibility measure: it encrypts the volume again, with the same 128 bit key and counter.

- Argue on whether the method provides a security margin which is larger, smaller or the same with respect to the original encryption scheme.
- Describe an alternative measure to comply with the directory, other than decrypting and re-encrypting the entire volume.



The compatibility measure is actually decrypting the volume, as applying twice the AES-CTR encryption function with the same key and counter adds via xor the same pseudorandom pad to the Ciphertext. The security margin is clearly lower than before: it's non-existent.

Encrypting with AES-CTR and a different 128 bit key actually solves the decryption issue, and provides 256 bits of equivalent security 

(under the largely believed assumption that AES is not a group).

Considering the aforementioned scenario, is it possible to claim that the information on the disk cannot be tampered with in a meaningful way, given that all the information on disk is fully encrypted? Either support the claim or disprove it providing a practical example and a solution to prevent tampering



Considering the aforementioned scenario, is it possible to claim that the information on the disk cannot be tampered with in a meaningful way, given that all the information on disk is fully encrypted? Either support the claim or disprove it providing a practical example and a solution to prevent tampering

Encrypting data with AES in counter mode does not provide any protection against tampering. Indeed, an attacker could modify the ciphertext at her own will, knowing that a bit flip in the ciphertext will result in a bit flip in the plaintext, in the same position.

Adding a message authentication code (MAC) to the data (e.g., disk-block-wise) prevents tampering altogether.

## Question 3

Consider the case of an air-gapped embedded device (i.e., without any network connection or similar communication means), where it is desired to have firmware updates performed connecting a mass storage device to it. The device does not have any permanent energy source (battery or connection to mains), and is thus turned off between uses.

Argue on whether it is possible or not to employ digital certificates and a PKI infrastructure to validate the origin authenticity of the firmware updates.

In case this is possible, sketch briefly the firmware validation procedure, otherwise highlight the reason why it cannot be performed, and provide a solution.

Consider the case of an air-gapped embedded device (i.e., without any network connection or similar communication means), where it is desired to have firmware updates performed connecting a mass storage device to it. The device does not have any permanent energy source (battery or connection to mains), and is thus turned off between uses. Argue on whether it is possible or not to employ digital certificates and a PKI infrastructure to validate the origin authenticity of the firmware updates. In case this is possible, sketch briefly the firmware validation procedure, otherwise highlight the reason why it cannot be performed, and provide a solution.

An air-gapped device without permanent power supply is not able to keep a reference to the passing of time, as a consequence, it is not possible to perform certificate validation due to the inability of testing for certificate expiration.

This shortcoming can either be circumvented switching to an alternate firmware validation approach (e.g., **employing a static public key to perform signature verification**), or obtaining a time reference. A viable approach to limit the connection of the device with the outer world is to decode the MSF-60 radio time reference, or to employ a GPS module. Note that, while at considerable effort, both can be spoofed.

A PIN selection form asks the user to pick a 6 decimal digits number to be employed to unlock a smartphone. Considering the case where a certain amount of users pick their birthdate, encoded as YYMMDD (least significant digits of the year, followed by month and day of birth) as their PIN, what is the average amount of attempts you have to make to guess one of their PINs?

Is it larger, smaller or about the same amount of attempts you have to make to guess a random sweep pattern over all the dots in a 2 x 4 grid?

A PIN selection form asks the user to pick a 6 decimal digits number to be employed to unlock a smartphone. Considering the case where a certain amount of users pick their birthdate, encoded as YYMMDD (least significant digits of the year, followed by month and day of birth) as their PIN, what is the average amount of attempts you have to make to guess one of their PINs?

Is it larger, smaller or about the same amount of attempts you have to make to guess a random sweep pattern over all the dots in a 2 x 4 grid?

A randomly chosen birthdate expressed as above is selected among  $365 \times 100$  possible ones (assuming that no clue about the age of the user is available). Ignoring the bias in the dates of birth (which are not uniformly distributed over the year), this implies that about 36500 guesses will yield a correct PIN, and, on average  $36500/2$  guesses will be sufficient.

A sweep pattern over a 4 x 2 grid is represented by the order in which the eight dots are connected in the sweep: this yields  $8!$ , that is 40320, possible sweep patterns, making a sweep pattern over a 4 x 2 grid only a little harder to guess than a random birthdate.

# Question 4

1.7. You (Eve) have intercepted two ciphertexts:

$c_1 = 1111100101111001110011000001011110000110$

$c_2 = 1111101001100111110111010000100110001000$

You know that both are OTP ciphertexts, encrypted with the *same key*. You know that **either**  $c_1$  is an encryption of **alpha** and  $c_2$  is an encryption of **bravo** **or**  $c_1$  is an encryption of **delta** and  $c_2$  is an encryption of **gamma** (all converted to binary from ASCII in the standard way).

Which of these two possibilities is correct, and why? What was the key  $k$ ?

1.7. You (Eve) have intercepted two ciphertexts:

$c_1 = 1111100101111001110011000001011110000110$

$c_2 = 1111101001100111110111010000100110001000$

You know that both are OTP ciphertexts, encrypted with the *same key*. You know that **either**  $c_1$  is an encryption of **alpha** and  $c_2$  is an encryption of **bravo** **or**  $c_1$  is an encryption of **delta** and  $c_2$  is an encryption of **gamma** (all converted to binary from ASCII in the standard way).

Which of these two possibilities is correct, and why? What was the key  $k$ ?

Observe that  $c_1 = p_1 \oplus k$  and  $c_2 = p_2 \oplus k$ . Adding memberwise the two equations will yield  $c_1 \oplus c_2 = p_1 \oplus p_2$ . Note that  $c_1 \oplus c_2$  is known to the attacker, and it is a bit string where a set bit in a given position tells the attacker that the bits in the corresponding positions of the plaintexts are different. Observe that the value of the xor of the last 8 bits of  $c_1$  and  $c_2$  is not all zeroes, therefore the last characters of the two plaintexts differ.

**We thus obtain that it's alpha and bravo**

## Question 5

Claim 1.3 For every  $m \in \{0, 1\}^\lambda$ , the distribution  $\text{EAVESDROP}(m)$  is the **uniform distribution** on  $\{0, 1\}^\lambda$ . Hence, for all  $m, m' \in \{0, 1\}^\lambda$ , the distributions  $\text{EAVESDROP}(m)$  and  $\text{EAVESDROP}(m')$  are identical.

1.9. Suppose we modify the subroutine discussed in Claim 1.3 so that it also returns  $k$ :

$\text{EAVESDROP}'(m \in \{0, 1\}^\lambda):$
$k \leftarrow \{0, 1\}^\lambda$
$c := k \oplus m$
return $(k, c)$

.

Is it still true that for every  $m$ , the output of  $\text{EAVESDROP}'(m)$  is distributed uniformly in  $(\{0, 1\}^\lambda)^2$ ? Or is the output distribution different for different choice of  $m$ ?



Claim 1.3 For every  $m \in \{0, 1\}^\lambda$ , the distribution  $\text{EAVESDROP}(m)$  is the **uniform distribution** on  $\{0, 1\}^\lambda$ . Hence, for all  $m, m' \in \{0, 1\}^\lambda$ , the distributions  $\text{EAVESDROP}(m)$  and  $\text{EAVESDROP}(m')$  are identical.

1.9. Suppose we modify the subroutine discussed in Claim 1.3 so that it also returns  $k$ :

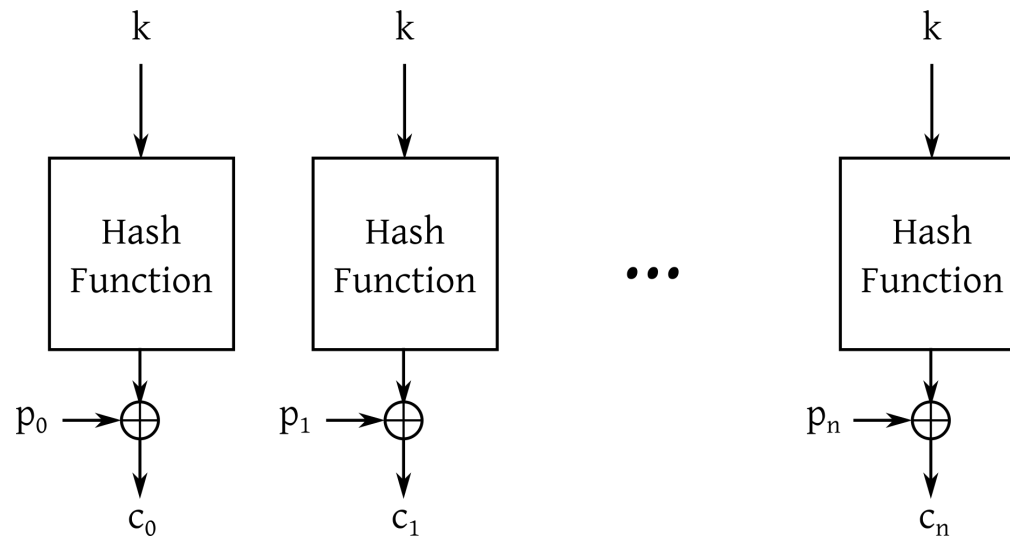
$\text{EAVESDROP}'(m \in \{0, 1\}^\lambda):$
$k \leftarrow \{0, 1\}^\lambda$
$c := k \oplus m$
return $(k, c)$

Is it still true that for every  $m$ , the output of  $\text{EAVESDROP}'(m)$  is distributed uniformly in  $(\{0, 1\}^\lambda)^2$ ? Or is the output distribution different for different choice of  $m$ ?

It's false. Simply consider the value obtained xor-ing the two returned values: it's  $k \oplus c = m$ . It's clear that the distribution of  $m$  does not look like a random independent bitstring with respect to  $m$ , indeed, it's the message!

## Question 6

Consider the following construction, where  $k$  is a 1024 bit long string, known only to Alice and Bob, who want to have confidential communications. The plaintext  $p$  is split in  $n+1$  blocks:  $p_0, p_1, \dots, p_n$ . Hash Function is a cryptographic hash function with a 512 bit digest. The ciphertext, which is sent on the line, is formed by the  $c_0, c_1, \dots, c_n$  blocks.



Is the communication with the depicted scheme (computationally) confidential against a passive attacker, who only eavesdrops on the ciphertext? If yes, argue why; if not, describe concisely what information is leaked, and propose a countermeasure to fix the information leaked.

No. Plaintext blocks with the same value will be encrypted to ciphertext blocks with the same value. Furthermore, if the attacker knows the position of a zero-filled plaintext block, she'll be able to decrypt all the messages sent.

Fix by appending a *nonce* to  $k$ , increase the *nonce* for each block.

The End

# Extra 1

- 2.4. Show that the following libraries are **not** interchangeable. Describe an explicit distinguishing calling program, and compute its output probabilities when linked to both libraries:

$\mathcal{L}_{\text{left}}$
$\text{EAVESDROP}(m_L, m_R \in \{\textcolor{red}{0}, \textcolor{red}{1}\}^\lambda):$
$k \leftarrow \{\textcolor{red}{0}, \textcolor{red}{1}\}^\lambda$
$c := k \oplus m_L$
return $(k, c)$

$\mathcal{L}_{\text{right}}$
$\text{EAVESDROP}(m_L, m_R \in \{\textcolor{red}{0}, \textcolor{red}{1}\}^\lambda):$
$k \leftarrow \{\textcolor{red}{0}, \textcolor{red}{1}\}^\lambda$
$c := k \oplus m_R$
return $(k, c)$

2.4. Show that the following libraries are **not** interchangeable. Describe an explicit distinguishing calling program, and compute its output probabilities when linked to both libraries:

$\mathcal{L}_{\text{left}}$
$\text{EAVESDROP}(m_L, m_R \in \{0, 1\}^\lambda):$ <hr/> $k \leftarrow \{0, 1\}^\lambda$ $c := k \oplus m_L$ return $(k, c)$

$\mathcal{L}_{\text{right}}$
$\text{EAVESDROP}(m_L, m_R \in \{0, 1\}^\lambda):$ <hr/> $k \leftarrow \{0, 1\}^\lambda$ $c := k \oplus m_R$ return $(k, c)$

```
tmp_k,tmp_c = eavesdrop(0^lambda,1^lambda)
```

```
if (tmp_k^tmp_c = 0^lambda)
```

```
    return "it's L_left!"
```

```
else
```

```
    return "it's L_right!"
```

The distinguisher works with  $\text{Pr} = 1$

- 5.12. Let  $G_1$  and  $G_2$  be deterministic functions, each accepting inputs of length  $\lambda$  and producing outputs of length  $3\lambda$ .
- (a) Define the function  $H(s_1 \| s_2) = G_1(s_1) \oplus G_2(s_2)$ . Prove that if **either** of  $G_1$  or  $G_2$  (or both) is a secure PRG, then so is  $H$ .
  - (b) What can you say about the simpler construction  $H(s) = G_1(s) \oplus G_2(s)$ , when one of  $G_1, G_2$  is a secure PRG?

### Definition

A CSPRNG is a deterministic function  $\text{PRNG}: \{0, 1\}^\lambda \rightarrow \{0, 1\}^{\lambda+l}$  whose output cannot be distinguished from an uniform random sampling of  $\{0, 1\}^{\lambda+l}$  in  $\mathcal{O}(\text{poly}(\lambda))$ .  $l$  is the CSPRNG stretch.

```

VERNAM( $\lambda$ ,  $\text{ptx} \in \{0, 1\}^\lambda$ )
-----
 $k \xleftarrow{\$} \{0, 1\}^\lambda$ 
 $\text{ctx} = \text{ptx} \oplus k$ 
return ctx

```

```

IDEAL( $\lambda$ ,  $\text{ptx} \in \{0, 1\}^\lambda$ )
-----
 $\text{ctx} \xleftarrow{\$} \{0, 1\}^\lambda$ 
return ctx

```

## HINTS

5.12. Let  $G_1$  and  $G_2$  be deterministic functions, each accepting inputs of length  $\lambda$  and producing outputs of length  $3\lambda$ .

- (a) Define the function  $H(s_1 \| s_2) = G_1(s_1) \oplus G_2(s_2)$ . Prove that if **either** of  $G_1$  or  $G_2$  (or both) is a secure PRG, then so is  $H$ .
- (b) What can you say about the simpler construction  $H(s) = G_1(s) \oplus G_2(s)$ , when one of  $G_1, G_2$  is a secure PRG?

a) Assume that  $G_1$  is the good PRG (the same reasoning applies if it's  $G_2$ ). If ~~the~~ the output of  $G_1$  cannot be told apart from a uniformly random sampled string of  $3\lambda$  bits, then the value  $G_1(s_1) \oplus G_2(s_2)$  is not distinguishable from random by Shannon's theorem (it's essentially a one time pad). As a consequence, the output of  $H$  is indistinguishable from a uniformly randomly chosen string.

b) You can reduce this case to the previous one. It's just feeding the previous  $H$  function with a non uniformly sampled string.



## Extra 3

11.13. Let  $H$  be a collision-resistant hash function with output length  $n$ . Let  $H^*$  denote iterating  $H$  in a manner similar to CBC-MAC:

$H^*(x_1 \cdots x_\ell)$ :

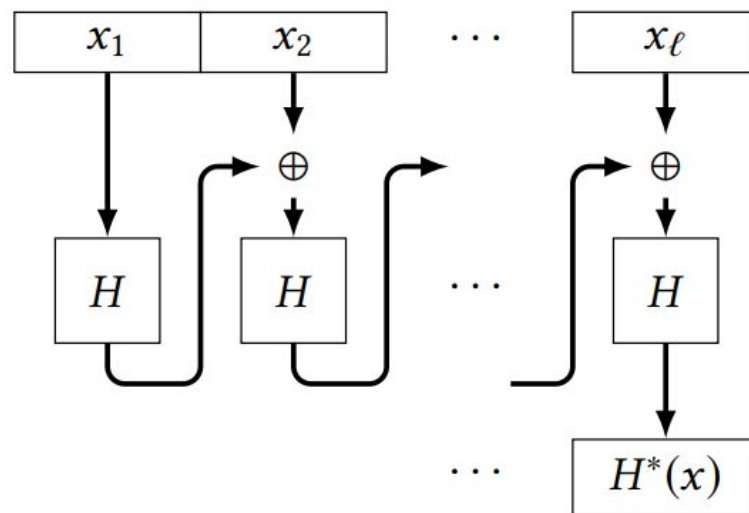
// each  $x_i$  is  $n$  bits

$y_0 := 0^n$

for  $i = 1$  to  $\ell$ :

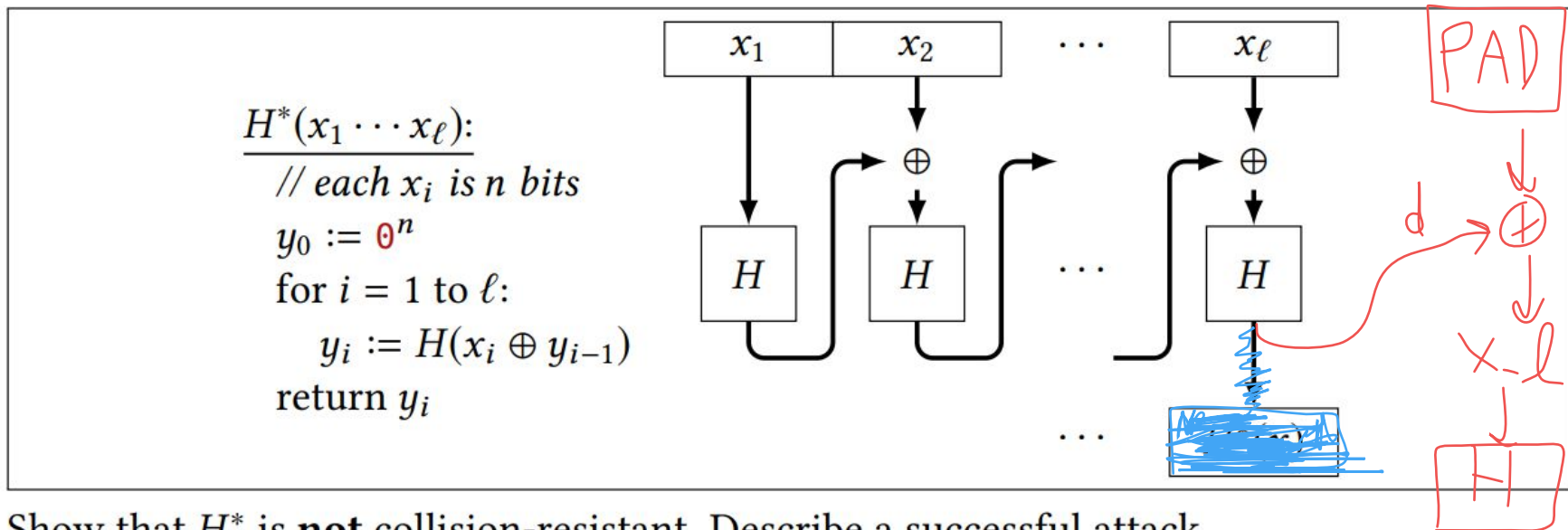
$y_i := H(x_i \oplus y_{i-1})$

return  $y_i$



Show that  $H^*$  is **not** collision-resistant. Describe a successful attack.

11.13. Let  $H$  be a collision-resistant hash function with output length  $n$ . Let  $H^*$  denote iterating  $H$  in a manner similar to CBC-MAC:



Take a random message  $x=(x_1, \dots, x_\ell)$ , compute the output of  $H^*$ , call it  $d$ . Compute  $\text{pad} = d \oplus x_\ell$ .  $H^*(x || \text{pad})$  will have the same hash as  $x$  since the last step will compute the hash of  $\text{pad} \oplus H^*(x) = d \oplus x_\ell \oplus H^*(x) = x_\ell$

## Extra 4

10.1. Consider the following MAC scheme, where  $F$  is a secure PRF with  $in = out = \lambda$ :

<u>KeyGen:</u> $k \leftarrow \{\mathbf{0}, \mathbf{1}\}^\lambda$ return $k$	<u><math>MAC(k, m_1 \  \dots \  m_\ell)</math>: // each <math>m_i</math> is <math>\lambda</math> bits</u> $m^* := \mathbf{0}^\lambda$ for $i = 1$ to $\ell$ : $m^* := m^* \oplus m_i$ return $F(k, m^*)$
---	--

Show that the scheme is **not** a secure MAC. Describe a distinguisher and compute its advantage.

10.1. Consider the following MAC scheme, where  $F$  is a secure PRF with  $in = out = \lambda$ :

<div> <div>KeyGen:</div> <div> <math>k \leftarrow \{0, 1\}^\lambda</math>  return <math>k</math> </div> </div>	<div> <div>MAC(<math>k, m_1 \parallel \dots \parallel m_\ell</math>): // each <math>m_i</math> is <math>\lambda</math> bits</div> <div> <math>m^* := 0^\lambda</math>  for <math>i = 1</math> to <math>\ell</math>:  <math>m^* := m^* \oplus m_i</math>  return <math>F(k, m^*)</math> </div> </div>
--	--

Show that the scheme is **not** a secure MAC. Describe a distinguisher and compute its advantage.

A secure MAC does not allow an attacker, who has seen a valid message-tag pair, to forge a valid tag for a different message. Assume you have a valid (m,t) pair. Denote with (m\_1,...,m\_l) the lambda bit long blocks of the message. Draw a lambda bits string at your leisure, call it pad.

The pair ( (m\_1^pad,m\_2^pad,...,m\_l), t) is a valid message-tag pair, where the message differs from m according to the pad you chose.