Cryptography

Academic Year 2025-2026

Homework 1

Francesco Testa, ID 1179083

October 16, 2025

Exercise 1.

Let's define the encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$. Because the length of message influences the length of the key, let's build the scheme using the length of the message as a subscript (i.e. $\Pi_n = (\mathsf{Gen}_n, \mathsf{Enc}_n, \mathsf{Dec}_n)$).
So let's define:

- Gen_n: return k, chosen from the set $\mathcal{K} = \{k \in \mathbb{N} | 1 < k < n\}$ The probability of output each integer key is $\frac{1}{|\mathcal{K}|} = \frac{1}{n-2}$.
- $\mathsf{Enc}_n(m, k)$: given a message

  $$m = \sigma_1 \cdot \sigma_2 \cdot ... \cdot \sigma_n \text{ where } \sigma_i \in \Sigma$$

  the algorithm builds a matrix of k columns and fills it by rows. If the message does not fit perfectly the matrix, the last row is filled with an extra padding character (i.e. $*, * \notin \Sigma$).
  Let's define r as the number of rows of the matrix: $r = \lceil \frac{n}{k} \rceil$.
  The matrix will be of size $r \times k$ and is defined as follows:

  $$\forall i, j | 1 \leq i \leq r, 1 \leq j \leq k \quad \mathcal{M}_{i,j} = \begin{cases} \sigma_{(i-1)*k+j} & \text{if } ((i-1)*k+j) \leq \ell, \\ * & \text{otherwise.} \end{cases}$$

  After the matrix is build, the ciphertext is obtained by reading the matrix by columns:

  $$c = \mathcal{M}_{1,1} \cdot \mathcal{M}_{2,1} \cdot \mathcal{M}_{r,1} \cdot \mathcal{M}_{1,2} \cdot ... \cdot \mathcal{M}_{r,k}$$

- $\mathsf{Dec}_n(c, k)$: given the ciphertext

  $$c = \sigma_1 \cdot \sigma_2 \cdot ... \cdot \sigma_l{}^1 \text{ where } \sigma_i \in \Sigma \cup \{*\}$$

  the algorithm builds a matrix of $\frac{l}{k}$ rows and k columns and fills it by columns. Let call r the number of rows of the matrix: $r = \frac{l}{k}$, where $l = |c|$. Note that $\frac{l}{k} = \lceil \frac{n}{k} \rceil$.
  So the matrix will be of size $r \times k$ and is defined as follows:

  $$\forall i, j | 1 \leq i \leq \frac{l}{k}, 1 \leq j \leq k \quad \mathcal{M}_{i,j} = \sigma_{(j-1)*r+i}$$

  From this matrix, the plaintext m' is obtained by reading the matrix by rows:

  $$m' = \mathcal{M}_{1,1} \cdot \mathcal{M}_{1,2} \cdot ... \cdot \mathcal{M}_{1,k} \cdot \mathcal{M}_{2,1} \cdot ... \cdot \mathcal{M}_{r,k}$$

  And finally, the algorithm returns m trimming the padding characters (i.e. $*$) from m'.

[1] The length of c can be different from n (i.e. $r \times k \geq n$). This is due to the padding characters added during the encryption phase.

Now, let's prove the **correctness** of Π.
We want to prove that $\mathsf{Dec}(\mathsf{Enc}(m,k),k) = m$. Let's call $\mathcal{M}_{\mathsf{Enc}}$ the matrix built during the encryption phase and $\mathcal{M}_{\mathsf{Dec}}$ the matrix built during the decryption phase. We can notice that:

$$\mathcal{M}_{\mathsf{Enc}} = \mathcal{M}_{\mathsf{Dec}} \implies \mathsf{Dec}(\mathsf{Enc}(m,k),k) = m$$

Called r the number of rows of both matrices ($r = \frac{l}{k} = \lceil \frac{n}{k} \rceil$), we want to prove that $\forall i,j | 1 \le i \le r, 1 \le j \le k$. $\mathcal{M}_{\mathsf{Enc}}[i,j] = \mathcal{M}_{\mathsf{Dec}}[i,j]$.
Because we read $\mathcal{M}_{\mathsf{Enc}}$ by columns to obtain c, we can say that, by definition, $c_{(j-1)r+i} = \mathcal{M}_{\mathsf{Enc}}[i,j]$. At the same time, $c_{(j-1)r+i} = \mathcal{M}_{\mathsf{Dec}}[i,j]$. So:

$$\mathcal{M}_{\mathsf{Enc}}[i,j] = c_{(j-1)r+i} = \mathcal{M}_{\mathsf{Dec}}[i,j]$$

And this proves the correctness of Π.
Now, let's prove that Π is **not perfectly-secure**.
First of all, $|\mathcal{K}| < |\mathcal{M}|$ because $|\mathcal{K}| = n-2$ and $|\mathcal{M}| = |\Sigma|^n$.
We can also design an adversary $\mathcal{A}$ such that:

$$Pr(\mathsf{PrivK}^{eav}_{\mathcal{A},\Pi_n} = 1) > \frac{1}{2}$$

Let's describe how $\mathcal{A}$ works:

- In the first phase, $\mathcal{A}$ chooses two messages m_0 and m_1 such that $m_0 = \alpha^n$ and $m_1 = \beta^n$, where $\alpha, \beta \in \Sigma$ and $\alpha \neq \beta$. Then, $\mathcal{A}$ sends m_0 and m_1 to the challenger.
- When the adversary receives the ciphertext c from the challenger, it checks if c contains the character β. If it does, then $\mathcal{A}$ outputs 1, otherwise it outputs 0.

In this way, we have that (the b value is the random bit chosen in the experiment):

$$\begin{aligned}
Pr(\mathsf{PrivK}^{eav}_{\mathcal{A},\Pi_n} = 1) =& Pr(A(c)=1|b=1)\cdot Pr(b=1) + Pr(A(c)=1|b=0)\cdot Pr(b=0) + \\
& Pr(A(c)=0|b=1)\cdot Pr(b=1) + Pr(A(c)=0|b=0)\cdot Pr(b=0) \\
=& 1\cdot\frac{1}{2} + 0\cdot\frac{1}{2} + 0\cdot\frac{1}{2} + 1\cdot\frac{1}{2} = \frac{1}{2}+\frac{1}{2} \\
=& 1 > \frac{1}{2}
\end{aligned}$$

So, we have that $Pr(\mathsf{PrivK}^{eav}_{\mathcal{A},\Pi_n} = 1) > \frac{1}{2}$, and this proves that Π_n is not perfectly-secure.
Because we have not used any specific instance of n in the proof, we can say that Π_n is not perfectly-secure for every n.

Exercise 2.
To prove that $H(x) = G(x)|_{\ell'(|x|)}$ we have to prove that:

- H stretches its input. By hypothesis, $\ell'(n) > n$, so using the input length as n, we have that $\ell'(|x|) > |x|$.
- H is polytime. This is true by definition: G is polytime and the truncation operation is polytime.
- H is pseudorandom. So, for every PPT algorithm D there exists $\epsilon \in \mathcal{NLG}$ such that: $|Pr(D(s)=1) - Pr(D(H(r))=1)| \le \epsilon(n)$, where $|s| = \ell'(n)$ and $|r| = n$.
  To prove this, we can use a proof by reduction.

  Let's assume that H is not pseudorandom, i.e.

$$|Pr(D_H(s)=1) - Pr(D_H(H(r))=1)| = \eta(n) \quad (\eta \text{ not negligible})$$

Algorithm 1 $D_G(x)$

```
1: $y \leftarrow x[1 .. \ell'(|x|)]$
2: return $D_H(y)$
```

This means that exists a Distinguisher D_H for H. Let's now use the Distinguisher D_H to build a Distinguisher D_G for G, as defined in Algorithm 1.

So, we have that:

$$Pr(D_G(G(x)) = 1) = 1$$
$$Pr(D_G(r) = 1) = \epsilon(n) \quad (\epsilon \text{ negligible})$$

So:

$$|Pr(D_G(G(x)) = 1) - Pr(D_G(r) = 1)| = 1 - \epsilon(n)$$

which is not negligible. This means that D_G is a Distinguisher for G, which is a contradiction. So, H is pseudorandom.

Exercise 3.
Let's analyze the following generators:

1. $G_1(x) = x \cdot 010$ is not a PRG.
G_1 stretches its input ($n + 3 > n$) and is polytime (it just appends 3 bits). But it is not pseudorandom. In a banal way, let's design a distinguisher for G_1 to prove this.

Algorithm 2 $D_{G_1}(x)$

```
1: y,z = split(x)    # $|y| = |x| - 3, |z| = 3$
2: if z==010 then
3:     return 1
4: end if
5: return 0
```

So:

$$|Pr(D_{G_1}(G_1(x)) = 1) - Pr(D_{G_1}(r) = 1)| = 1 - \frac{2^{n-3}}{2^n} = 1 - \frac{1}{8} = \frac{7}{8}$$

which is not negligible. So G_1 is not a PRG.

2. $G_2(x) = F(x, 0^{|x|})$ is not a PRG.
The definition of a pseudorandom function states that the binary partial function must be length-preserving. In this case F is defined as follows: $F : \{0,1\}^{|x|} \times \{0,1\}^{|x|} \rightarrow \{0,1\}^{|x|}$. The output length is $|x|$, that is the same of G_2, therefore G_2 does not stretch its input. So, it is not a PRG.

3. G_3 is defined as follows:

$$G_3(x) = \begin{cases} x \cdot x & \text{if } |x| \leq 2, \\ F(x, 1^{|x|}) \cdot F(x, 0^{|x|}) & \text{otherwise.} \end{cases}$$

Let's prove that G_3 is a PRG:

- G_3 stretches its input. In fact, let's call n the length of x ($n = |x|$). In both cases, the length of the output is $2n > n$. In particular, this holds because in the first case there is only the concatenation of the same x, while in the second case we know that F is length-preserving by definition.

- G_3 is polytime. This is true in both cases: F is polytime and the concatenation operation is polytime.
- G_3 is pseudorandom. So, for every PPT algorithm D there exists $\epsilon \in \mathcal{NLG}$ such that: $|Pr(D(s) = 1) - Pr(D(G_3(r)) = 1)| \leq \epsilon(n)$, where $|s| = 2n$ and $|r| = n$.
  Let's analyze $G'_3 = F(x, 1^{|x|}) \cdot F(x, 0^{|x|})$. We can prove that G'_3 is a PRG by reduction. Let's assume that G'_3 is not a PRG, so exists the distinguisher $D_{G'_3}$ such that:
  $$|Pr(D_{G'_3}(G'_3(x)) = 1) - Pr(D_{G'_3}(r) = 1)| = \eta(n) \quad (\eta \text{ not negligible})$$
  We can build a distinguisher D_F for F as in Algorithm 3. We can assume that the distinguisher has access to an oracle O that can be either $F_k(\cdot)$ or $f(\cdot)$.

Algorithm 3 $D_F(1^n)$

```
1: $y \leftarrow O(0^n)$
2: $z \leftarrow O(1)^n$
3: return $D_{G'_3}(y \cdot z)$
```

So, for how the distinguisher is built, we have that:
$$Pr(D_F^{F_k(\cdot)}(1^n) = 1) = Pr(D_{G'_3}(G'_3(x)) = 1)$$
$$Pr(D_F^{f(\cdot)}(1^n) = 1) = Pr(D_{G'_3}(r) = 1)$$
This implies that:
$$\begin{aligned}|Pr(D_F^{F_k(\cdot)}(1^n) = 1) - Pr(D_F^{f(\cdot)}(1^n) = 1)| &= |Pr(D_{G'_3}(G'_3(x)) = 1) - Pr(D_{G'_3}(r) = 1)| \\ &= \eta(n)\end{aligned}$$
which is not negligible. This means that D_F is a distinguisher for F, which is a contradiction. So, G'_3 is a PRG.
Finally, let's prove that G_3 is a PRG. To prove that, we can notice that the definition of PRG is asymptotic, so we can ignore the case in which $G_3(x) = x \cdot x$, because it happens only for $|x| \leq 2$.
So we can assume that $G_3(x) = G'_3(x)$ and therefore we can prove it by reduction and define a distinguisher D_{G_3} fot G_3 that is equal to $D_{G'_3}$ used before for G'_3. analyzing the probability we have that:
$$\begin{aligned}|Pr(D_{G_3}(G_3(x)) = 1) - Pr(D_{G_3}(r) = 1)| &\approx |Pr(D_{G'_3}(G'_3(x)) = 1) - Pr(D_{G'_3}(r) = 1)| \\ &\approx \eta(n) \quad (\eta \text{ not negligible})\end{aligned}$$
So this means that D_{G_3} is a distinguisher for G_3, which is a contradiction. So, G_3 is a PRG.

Now let's analyze the following functions:

1. $F_1(k, x) = x \oplus k$. Let's prove that F_1 is not a PRF.
   By definition, F_1 is a PRF if:
   - F_1 is length-preserving. This is true because $|F_1(k, x)| = |x \oplus k| = |x| = |k|$.
   - F_1 is efficient. This is true because the XOR operation is obviously polytime.
   - F_1 is pseudorandom. This is not true: we can build a distinguisher D_{F_1} as in Algorithm 4 that distinguishes F_1 from a truly random function. The idea is to exploit the property of XOR: $a \oplus b \oplus b = a$.
     When oracle uses f, then y and z are random strings and independent. So, analyzing the probability:
     $$|Pr(D_{F_1}^{F_1(k,\cdot)}(1^n) = 1) - Pr(D_{F_1}^{f(\cdot)}(1^n) = 1)| = 1 - \frac{1}{2^n}$$

Algorithm 4 $D_{F_1}(1^n)$ # We have access to Oracle O for F_1 or f

```
y ← O(0^n)
z ← O(1^n)
w ← z ⊕ 1^n
if y==w then
    return 1
end if
return 0
```

which is not negligible. So, contradicting the hypothesis we have that F_1 is not a PRF.

2. $F_2(k, m) = G(k)|_{|k|} \oplus m$. Let's prove that F_2 is not a PRF.
   By definition, F_2 is a PRF if:
   - F_2 is length-preserving. This is true because $|F_2(k, m)| = |G(k)|_{|k|} \oplus m| = |m| = |G(k)|_{|k|}|$.
   - F_2 is efficient. This is true because by hypothesis G is a PRG and by definition of PRG G is polytime; the XOR operation also is polytime.
   - F_2 is pseudorandom. This is not true: we can build a distringuisher D_{F_2} as in Algorithm 5 that distinguishes F_2 from a truly random function. The idea is the same used for F_1.

Algorithm 5 $D_{F_2}(1^n)$ # We have access to Oracle O for F_2 or f

```
y ← O(0^n)
z ← O(1^n)
w ← z ⊕ 1^n
if y==w then
    return 1
end if
return 0
```

This distinguisher works because the pseudorandom generator G is called with the same key k in both calls, and due to the deterministic property of PRGs, $G(k)|_{|k|}$ is the same. In fact, when the oracle uses the F_2, the k value is chosen randomly once and then fixed. So, exploiting the property of XOR: $a \oplus b \oplus b = a$ as in previous exercise, we can build the distringuisher, and analyzing the probability we have that:

$$|Pr(D_{F_2}^{F_2(k,\cdot)}(1^n) = 1) - Pr(D_{F_2}^{f(\cdot)}(1^n) = 1)| = 1 - \frac{1}{2^n}$$

which is not negligible. So the proof of this second exercise is equal to the proof of the first exercise, and contraddicting the hypothesis we have that F_2 is not a PRF.