Examen

Sécurité et cryptographie

Classes: 2ère année ingénieur GTR

Session principale

Enseignant : M. H. Hdhili

Documents autorisés

Exercice 1 [5pts]:

1) Soit le programme PROG1 (donné en annexe) implémentant une technique de scan de port. Répondre aux questions suivantes :
   a. De quel type de scan de port s'agit-il ? Expliquer ?
   b. Donner le nombre de paquets échangés entre la machine qui lance le scan et la machine scanné pour un seul port considéré fermé.
   c. Ce programme permet-il de scanner tous les ports possibles ($0 \rightarrow 2^{16}-1$)? Expliquer ?

2) Soit la capture de trames CATURE1 (donnée en annexe) faite au cours de l'exécution d'un programme de scan de ports. Répondre aux questions suivantes :
   a. Donner un exemple d'un ensemble de paquets correspondant à un port fermé
   b. Donner un exemple d'un ensemble de paquets correspondant à un port ouvert
   c. De quel type de scan s'agit-il ? Expliquer ?

3) Considérer le « TCP ACK scan » pour répondre aux questions suivantes :
   a. Dans le cas normal, quel est la valeur du bit ACK du premier paquet envoyé d'un client vers un serveur ? Expliquer ?
   b. Dans le cas du « TCP ACK scan », Quel est la valeur du bit ACK envoyé dans le premier paquet vers la machine scanné ?
   c. Expliquer alors comment la machine qui effectue le scan détermine que les ports qu'elle est en train de scanner sont filtré ou non ?

Exercice 2 [7pts]:

1) Quel est la principale différence entre le sniffing actif et le sniffing passif ?
2) Est ce que le sniffing actif est utile pour les réseaux Ethernet partagé ? Expliquer ?
3) Considérer le programme PROG2 implémentant un simple sniffer analyseur de trames pour répondre aux questions suivantes :
   a. S'agit-il d'un sniffer actif ou passif ? Expliquer ?
   b. A quoi sert la fonction « **BindRawSocketToInterface » ?**
   c. Donner le nom de la fonction ou la ligne de code permettant de récupérer la trame sous son format brut
   d. Préciser les protocoles niveau liaison, réseau et transport que ce code permet d'analyser

e. Augmenter ce code en proposant une fonction permettant d'analyser les datagrammes UDP définie avec la variable **IPPROTO_UDP** et dont la structure est décrite dans « udp.h » de la façon suivante :

```
Struct udphdr
{
      u_int16_t  source ; // port source
      u_int16_t  dest ;/ /port destination
     u_int16_t  len ; // longueur
     u_int16_t  check ; // checksum
}
```

Exercice 3 [8pts]:

Soit **M** un message divisé en blocs $\{x_1,x_2,x_3,\ldots x_p\}$chacun de taille **n** bits et soit **K** une clé de même taille que les blocs (n bits). Soit $\{c_1,c_2,c_3,\ldots c_p\}$les cryptogrammes des blocs obtenus en appliquant la clé K aux blocs. Le chiffrement des blocs se fait selon le schéma suivant:

C_0= IV (valeur initiale) ; pour i de 1 à p, $c_j=E_K(C_{j-1}\oplus x_j)$

1) La fonction E_K est inversible et son inverse est $\mathbf{D_K}$. Montrer que l'opération de déchiffrement est **$x_j=C_{j-1}\oplus D_K(C_j)$** (rappel : $A\oplus A=0$; $A\oplus 0=A$, $A\oplus B= B\oplus A$)
2) Peut-on chiffrer un bloc quelconque du message M sans chiffrer les blocs qui le précèdent ? Expliquer ?
3) Peut-on déchiffrer un bloc quelconque c_i sans déchiffrer les blocs qui le précèdent ? Expliquer ?
4) Peut-on déchiffrer un bloc $\mathbf{c_j}$ en l'absence des autres blocs chiffrés ? Expliquer ?
5) Prenons le cas où **$E_K(x)=D_K(x)=K\oplus x$.** Supposons qu'un attaquant a pu récupérer deux blocs consécutifs (x_{j-1},x_j) ainsi que leurs cryptogrammes correspondants (c_{j-1},c_j). Montrer que cet attaquant peut en déduire la clé de chiffrement K.
6) Soient A et B deux entités utilisant le procédé de chiffrement décrit dans cet exercice. La clé K doit être échangée d'une façon **sécurisé et authentifié**. Pour cela A et B font appel au chiffrement asymétrique. A calcule la clé K, la chiffre pour obtenir KC et l'envoi à B.
    a. Avec quelle clé A doit chiffrer K ?
    b. Avec quelle clé B déchiffre KC ?
    c. Expliquer pourquoi cette méthode n'est pas authentifiée et proposer une solution ?

Annexe :

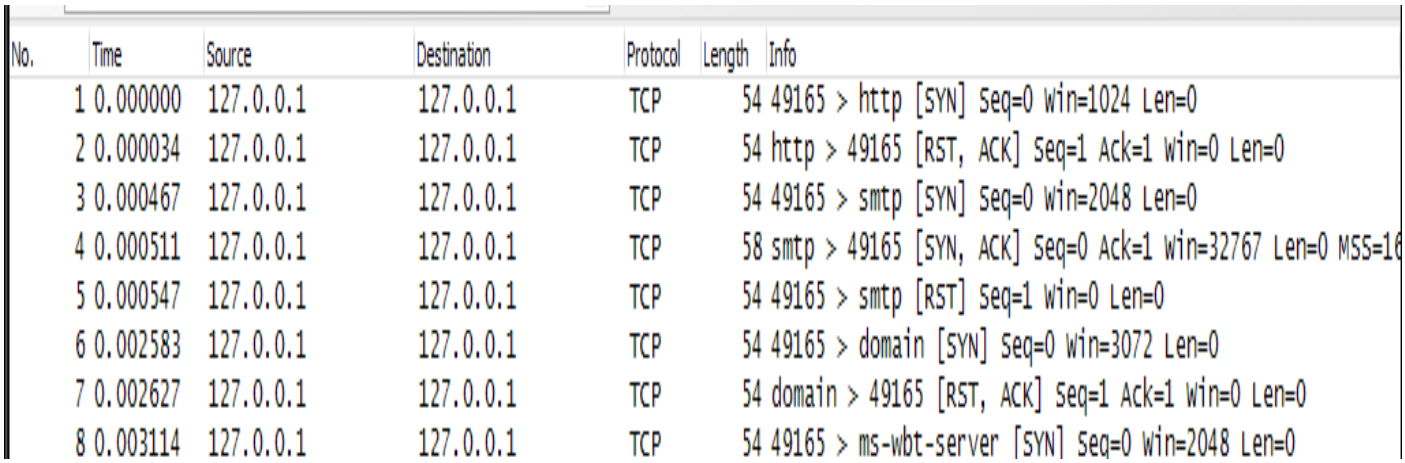
PROG1 :

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>

/* Main programs starts*/
int main(int argc, char **argv)
{
     int sd; //socket descriptor
     int port; //port number
     int start; //start port
     int end; //end port
     int rval; //socket descriptor for connect
     char responce[1024]; //to receive data
     char *message="shell"; //data to send
     struct hostent *hostaddr; //To be used for IPaddress
     struct sockaddr_in servaddr; //socket structure

     if (argc < 4 )
     {
          printf("Usage: ./tscan <IPaddress> <Start Port> <End Port>\n");
          return (EINVAL);
     }
     start = atoi(argv[2]); //portno is ascii to int second argument
     end = atoi(argv[3]);
     for (port=start; port<=end; port++)
     {
          sd = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP); //created the tcp socket
          if (sd == -1)
         {
               perror("Socket()\n");
               return (errno);
          }
          memset( &servaddr, 0, sizeof(servaddr));
          servaddr.sin_family = AF_INET;
          servaddr.sin_port = htons(port); //set the portno
          hostaddr = gethostbyname( argv[1] ); //get the ip 1st argument
          memcpy(&servaddr.sin_addr, hostaddr->h_addr, hostaddr->h_length);
         //below connects to the specified ip in hostaddr
         rval = connect(sd, (struct sockaddr *) &servaddr, sizeof(servaddr));
         if (rval != -1)
        {
            printf("Port %d is open\n",port);
        }
        close(sd); //socket descriptor
     }
}
```

CATURE1 :



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	54	49165 > http [SYN] Seq=0 Win=1024 Len=0
2	0.000034	127.0.0.1	127.0.0.1	TCP	54	http > 49165 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
3	0.000467	127.0.0.1	127.0.0.1	TCP	54	49165 > smtp [SYN] Seq=0 Win=2048 Len=0
4	0.000511	127.0.0.1	127.0.0.1	TCP	58	smtp > 49165 [SYN, ACK] Seq=0 Ack=1 Win=32767 Len=0 MSS=16
5	0.000547	127.0.0.1	127.0.0.1	TCP	54	49165 > smtp [RST] Seq=1 Win=0 Len=0
6	0.002583	127.0.0.1	127.0.0.1	TCP	54	49165 > domain [SYN] Seq=0 Win=3072 Len=0
7	0.002627	127.0.0.1	127.0.0.1	TCP	54	domain > 49165 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
8	0.003114	127.0.0.1	127.0.0.1	TCP	54	49165 > ms-wbt-server [SYN] Seq=0 Win=2048 Len=0

PROG2 :

```
#include <stdio.h>
#include <sys/types.h>
#include<stdio.h>
#include<stdlib.h>
#include<sys/socket.h>
#include<features.h>
#include<linux/if_packet.h>
#include<linux/if_ether.h>
#include<errno.h>
#include<sys/ioctl.h>
#include<net/if.h>
#include <netinet/ip.h>
#include <netinet/tcp.h>
#include <string.h>
#include<netinet/in.h>

int CreateRawSocket(int protocol_to_sniff)
{
    int rawsock;
    if((rawsock = socket(PF_PACKET, SOCK_RAW, htons(protocol_to_sniff)))== -1)
    {
            perror("Error creating raw socket: ");
            exit(-1);
    }
    return rawsock;
}
int BindRawSocketToInterface(char *device, int rawsock, int protocol)
{
    struct sockaddr_ll sll;
    struct ifreq ifr;
    bzero(&sll, sizeof(sll));
    bzero(&ifr, sizeof(ifr));
    /* First Get the Interface Index  */
    strncpy((char *)ifr.ifr_name, device, IFNAMSIZ);
    if((ioctl(rawsock, SIOCGIFINDEX, &ifr)) == -1)
    {
            printf("Error getting Interface index !\n");
            exit(-1);
    }
    /* Bind our raw socket to this interface */
    sll.sll_family = AF_PACKET;
```

```
	sll.sll_ifindex = ifr.ifr_ifindex;
	sll.sll_protocol = htons(protocol);

	if((bind(rawsock, (struct sockaddr *)&sll, sizeof(sll)))== -1)
	{
		perror("Error binding raw socket to interface\n");
		exit(-1);
	}
	return 1;
}
void PrintPacketInHex(unsigned char *packet, int len)
{
	unsigned char *p = packet;
	printf("\n\n---------Packet---Starts----\n\n");
	while(len--)
	{
		printf("%.2x ", *p);
		p++;
	}
	printf("\n\n--------Packet---Ends-----\n\n");
}
PrintInHex(char *mesg, unsigned char *p, int len)
{
	printf(mesg);
	while(len--)
	{
		printf("%.2X ", *p);
//		printf("%c",*p);
		p++;
	}
}
ParseEthernetHeader(unsigned char *packet, int len)
{
	struct ethhdr *ethernet_header;
	if(len > sizeof(struct ethhdr))
	{
		ethernet_header = (struct ethhdr *)packet;
		/* First set of 6 bytes are Destination MAC */
		PrintInHex("Destination MAC: ", ethernet_header->h_dest, 6);
		printf("\n");
		/* Second set of 6 bytes are Source MAC */
		PrintInHex("Source MAC: ", ethernet_header->h_source, 6);
		printf("\n");
		/* Last 2 bytes in the Ethernet header are the protocol it carries */
		PrintInHex("Protocol: ",(void *)&ethernet_header->h_proto, 2);
		printf("\n");
	}
	else
	{
		printf("Packet size too small !\n");
	}
}
ParseIpHeader(unsigned char *packet, int len)
{
	struct ethhdr *ethernet_header;
	struct iphdr *ip_header;
```

```
    /* First Check if the packet contains an IP header using
       the Ethernet header                                  */
    ethernet_header = (struct ethhdr *)packet;
    if(ntohs(ethernet_header->h_proto) == ETH_P_IP)
    {
            /* The IP header is after the Ethernet header  */
            if(len >= (sizeof(struct ethhdr) + sizeof(struct iphdr)))
            {
                    ip_header = (struct iphdr*)(packet + sizeof(struct ethhdr));
                    /* print the Source and Destination IP address */
                    printf("Dest IP address: %s\n", inet_ntoa(ip_header->daddr));
                    printf("Source IP address: %s\n", inet_ntoa(ip_header->saddr));
            }
            else
            {
                    printf("IP packet does not have full header\n");
            }
    }
    else
    {
            /* Not an IP packet */
    }
}
ParseTcpHeader(unsigned char *packet , int len)
{
    struct ethhdr *ethernet_header;
    struct iphdr *ip_header;
    struct tcphdr *tcp_header;
    /* Check if enough bytes are there for TCP Header */
    if(len >= (sizeof(struct ethhdr) + sizeof(struct iphdr) + sizeof(struct tcphdr)))
    {
            /* Do all the checks: 1. Is it an IP pkt ? 2. is it TCP ? */
            ethernet_header = (struct ethhdr *)packet;
            if(ntohs(ethernet_header->h_proto) == ETH_P_IP)
            {
                    ip_header = (struct iphdr *)(packet + sizeof(struct ethhdr));
                    if(ip_header->protocol == IPPROTO_TCP)
                    {
                            tcp_header = (struct tcphdr*)(packet + sizeof(struct ethhdr) +
ip_header->ihl*4 );
                            /* Print the Dest and Src ports */
                            printf("Source Port: %d\n", ntohs(tcp_header->source));
                            printf("Dest Port: %d\n", ntohs(tcp_header->dest));
                    }
                    else
                    {
                            printf("Not a TCP packet\n");
                    }
            }
            else
            {
                    printf("Not an IP packet\n");
            }

    }
    else
```

```
    {
            printf("TCP Header not present \n");

    }
}
int ParseData(unsigned char *packet, int len)
{
    struct ethhdr *ethernet_header;
    struct iphdr *ip_header;
    struct tcphdr *tcp_header;
    unsigned char *data;
    int data_len;
    /* Check if any data is there */
    if(len > (sizeof(struct ethhdr) + sizeof(struct iphdr) + sizeof(struct tcphdr)))
    {

            ip_header = (struct iphdr*)(packet + sizeof(struct ethhdr));
            data = (packet + sizeof(struct ethhdr) + ip_header->ihl*4 +sizeof(struct tcphdr));
            data_len = ntohs(ip_header->tot_len) - ip_header->ihl*4 - sizeof(struct tcphdr);

            if(data_len)
            {
                    printf("Data Len : %d\n", data_len);
                    printf("----------\n");
                    printf("%s\n",(char*)data);
                    printf("**********\n");
                    PrintInHex("Data : ", data, data_len);
                    printf("\n\n");
                    return 1;
            }
            else
            {
                    printf("No Data in packet\n");
                    return 0;
            }
    }
    else
    {
            printf("No Data in packet\n");
            return 0;
    }

}
int IsIpAndTcpPacket(unsigned char *packet, int len)
{
    struct ethhdr *ethernet_header;
    struct iphdr *ip_header;
    ethernet_header = (struct ethhdr *)packet;
    if(ntohs(ethernet_header->h_proto) == ETH_P_IP)
    {
            ip_header = (struct iphdr *)(packet + sizeof(struct ethhdr));
            if(ip_header->protocol == IPPROTO_TCP)
                    return 1;
            else
                    return -1;
    }
```

```
	else
	{
		return -1;
	}

}
main(int argc, char **argv)
{
	int raw;
	unsigned char packet_buffer[2048];
	int len;
	int packets_to_sniff;
	struct sockaddr_ll packet_info;
	int packet_info_size = sizeof(packet_info);

	if (argc < 3 )
	{
		printf("-------------------------------------------------\n");
		printf("Usage: ./sniffer <Interface> <Nbre of packets to sniff>\n");
		printf("-------------------------------------------------\n");
		return (EINVAL);
	}

	/* create the raw socket */
	raw = CreateRawSocket(ETH_P_IP);
	/* Bind socket to interface */
	BindRawSocketToInterface(argv[1], raw, ETH_P_IP);
	/* Get number of packets to sniff from user */
	packets_to_sniff = atoi(argv[2]);
	/* Start Sniffing and print Hex of every packet */
	while(packets_to_sniff--)
	{
		if((len = recvfrom(raw, packet_buffer, 2048, 0, (struct sockaddr*)&packet_info,
(socklen_t*)&packet_info_size)) == -1)
		{
			perror("Recv from returned -1: ");
			exit(-1);
		}
		else
		{
			/* Packet has been received successfully !! */
			PrintPacketInHex(packet_buffer, len);
			ParseEthernetHeader(packet_buffer, len);
			ParseIpHeader(packet_buffer, len);
			ParseTcpHeader(packet_buffer, len);
			if(IsIpAndTcpPacket(packet_buffer, len))
			{
				if(!ParseData(packet_buffer, len))
					packets_to_sniff++;
			}
		}
	}
	return 0;
}
```

Correction :Examen

Sécurité et cryptographie

Classes: 2ère année ingénieur GTR

Session principale

Enseignant : M. H. Hdhili

Documents autorisés

Exercice 1 [5pts]:

4) Soit le programme PROG1 (donné en annexe) implémentant une technique de scan de port. Répondre aux questions suivantes :
   a. De quel type de scan de port s'agit-il ? Expliquer ?
      ⇨ **TCP connect scan puisqu'il utilise l'appel de la fonction *connect***
   b. Donner le nombre de paquets échangés entre la machine qui lance le scan et la machine scanné pour un seul port considéré fermé.
      ⇨ **2 paquets (1 paquet SYN et 1 paquet (RST, ACK))**
   c. Ce programme permet-il de scanner tous les ports possibles ($0 \rightarrow 2^{16}-1$)? Expliquer ?
      ⇨ **Oui à l'exécution, nous écrivons . *./tscan 0 $2^{16}-1$***

5) Soit la capture de trames CATURE1 (donnée en annexe) faite au cours de l'exécution d'un programme de scan de ports. Répondre aux questions suivantes :
   a. Donner un exemple d'un ensemble de paquets correspondant à un port fermé
      ⇨ **Paquet 1 et 2 ou 6 et 7**
   b. Donner un exemple d'un ensemble de paquets correspondant à un port ouvert
      ⇨ **Paquet 3,4 et 5**
   c. De quel type de scan s'agit-il ? Expliquer ?
      ⇨ **TCP SYN scan (half open scan) car le scanneur envoi un RST en cas de succès**

6) Considérer le « TCP ACK scan » pour répondre aux questions suivantes :
   a. Dans le cas normal, quel est la valeur du bit ACK du premier paquet envoyé d'un client vers un serveur ? Expliquer ?
      ⇨ **ACK=0 puisque le client n'a encore rien reçu à partir du serveur. Donc, il ne va rien acquitter**
   b. Dans le cas du « TCP ACK scan », Quel est la valeur du bit ACK envoyé dans le premier paquet vers la machine scanné ?
      ⇨ **ACK=1**
   c. Expliquer alors comment la machine qui effectue le scan détermine que les ports qu'elle est en train de scanner sont filtré ou non ?
      ⇨ **ACK=1 dans le premier paquet va engendrer le rejet du paquet correspondant s'il sera traité par un firewall statefull**

Exercice 2 [7pts]:

4) Quel est la principale différence entre le sniffing actif et le sniffing passif ?

   ⇨ **passif : écoute du réseau sans injection de paquets**

   ⇨ **actif : injection de paquets (exécution d'attaques…) pour pouvoir écouter le trafic réseau**

5) Est ce que le sniffing actif est utile pour les réseaux Ethernet partagé ? Expliquer ?

   ⇨ **Non puisque les paquets sont automatiquement diffusés dans le réseau et le sniffer va donc recevoir une copie de ces paquets**

6) Considérer le programme PROG2 implémentant un simple sniffer analyseur de trames pour répondre aux questions suivantes :

   a. S'agit-il d'un sniffer actif ou passif ? Expliquer ?

   ⇨ **Sniffer passif : il n'y a aucune fonction (send, Write,…) permettant d'injecter/envoyer des paquets**

   b. A quoi sert la fonction « **BindRawSocketToInterface » ?**

   ⇨ **Choix de l'interface à écouter**

   c. Donner le nom de la fonction ou la ligne de code permettant de récupérer la trame sous son format brut

   ⇨ **Function recvfrom**

   d. Préciser les protocoles niveau liaison, réseau et transport que ce code permet d'analyser

   ⇨ **Liaison : Ethernet. Réseau : IP. Transport : TCP**

   e. Augmenter ce code en proposant une fonction permettant d'analyser les datagrammes UDP définie avec la variable **IPPROTO_UDP** et dont la structure est décrite dans « udp.h » de la façon suivante :

```
Struct udphdr
{
     u_int16_t  source ; // port source
     u_int16_t  dest ;/ /port destination
    u_int16_t  len ; // longueur
    u_int16_t  check ; // checksum
}
```

   ⇨ Il faut copier le code de la fonction **ParseTcpHeader(unsigned char *packet , int len)** puis le modifier telle que c'est indiqué en gras dans la réponse suivante

```
ParseUdpHeader(unsigned char *packet , int len)
{
    struct ethhdr *ethernet_header;
    struct iphdr *ip_header;
    struct udphdr *udp_header;
    /* Check if enough bytes are there for UDP Header */
    if(len >= (sizeof(struct ethhdr) + sizeof(struct iphdr) + sizeof(struct udphdr)))
    {
            /* Do all the checks: 1. Is it an IP pkt ? 2. is it udp ? */
            ethernet_header = (struct ethhdr *)packet;
            if(ntohs(ethernet_header->h_proto) == ETH_P_IP)
            {
                    ip_header = (struct iphdr *)(packet + sizeof(struct ethhdr));
                    if(ip_header->protocol == IPPROTO_UDP)
                    {
                            udp_header = (struct udphdr*)(packet + sizeof(struct ethhdr) +
ip_header->ihl*4 );
                            /* Print the Dest and Src ports */
                            printf("Source Port: %d\n", ntohs(udp_header->source));
```

```
                        printf("Dest Port: %d\n", ntohs(udp_header->dest));
                }
                else
                {
                        printf("Not an UDP packet\n");
                }
        }
        else
        {       printf("Not an IP packet\n");       }
    }
    else
    {       printf("UDP Header not present \n");

    }
}
```

Exercice 3 [8pts]:

Soit **M** un message divisé en blocs $\{x_1,x_2,x_3,\ldots x_p\}$chacun de taille **n** bits et soit **K** une clé de même taille que les blocs (n bits). Soit $\{c_1,c_2,c_3,\ldots c_p\}$les cryptogrammes des blocs obtenus en appliquant la clé K aux blocs. Le chiffrement des blocs se fait selon le schéma suivant:

C_0= IV (valeur initiale) ; pour i de 1 à p, $c_j=E_K(C_{j-1}\oplus x_j)$

1) La fonction E_K est inversible et son inverse est **D_K**. Montrer que l'opération de déchiffrement est **$x_j=C_{j-1}\oplus D_K(C_j)$** (rappel : $A\oplus A=0$; $A\oplus 0=A$, $A\oplus B=B\oplus A$)

$c_j=E_K(C_{j-1}\oplus x_j)$ ➔ $D_K(c_j)=D_K(E_K(C_{j-1}\oplus x_j))$
➔ $D_K(c_j)=C_{j-1}\oplus x_j$
➔ $C_{j-1}\oplus D_K(c_j)=C_{j-1}\oplus C_{j-1}\oplus x_j$
➔ $C_{j-1}\oplus D_K(c_j)=x_j$

2) Peut-on chiffrer un bloc quelconque du message M sans chiffrer les blocs qui le précèdent ? Expliquer ?
   ➔ Non, selon la formule $x_j=C_{j-1}\oplus D_K(C_j)$, le chiffrement de x_j nécessite C_{j-1}
3) Peut-on déchiffrer un bloc quelconque c_i sans déchiffrer les blocs qui le précèdent ? Expliquer ?
   ➔ Oui, x_j ne dépend pas de x_{j-1}
4) Peut-on déchiffrer un bloc **c_j** en l'absence des autres blocs chiffrés ? Expliquer ?
   ➔ Non, selon la formule $x_j=C_{j-1}\oplus D_K(C_j)$, le déchiffrement de c_j nécessite C_{j-1}
5) Prenons le cas où **$E_K(x)=D_K(x)=K\oplus x$.** Supposons qu'un attaquant a pu récupérer deux blocs consécutifs (x_{j-1},x_j) ainsi que leurs cryptogrammes correspondants (c_{j-1},c_j). Montrer que cet attaquant peut en déduire la clé de chiffrement K.
   Dans ce cas : $c_j=K\oplus C_{j-1}\oplus x_j$ ➔ $K=c_j\oplus C_{j-1}\oplus x_j$
6) Soient A et B deux entités utilisant le procédé de chiffrement décrit dans cet exercice. La clé K doit être échangée d'une façon **sécurisé et authentifié**. Pour cela A et B font appel au chiffrement asymétrique. A calcule la clé K, la chiffre pour obtenir KC et l'envoi à B.
   a. Avec quelle clé A doit chiffrer K ? ➔ **avec la clé publique de B**
   b. Avec quelle clé B déchiffre KC ? ➔**avec sa clé privée**
   c. Expliquer pourquoi cette méthode n'est pas authentifiée et proposer une solution ?
      ➔ Rien ne garantit l'appartenance de la clé publique de B à B.
      Solution : certification de la clé publique de B.