II4021 Kriptografi

12- Kriptografi Modern

Oleh: Rinaldi M

Program Studi Sistem dan Teknologi Informasi
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2026

Pendahuluan

- Kriptografi modern adalah era kriptografi setelah penemuan komputer digital.
- Perkembangan teknologi komputer digital membuat ilmu kriptografi berkembang dengan pesat.
- Komputer digital merepresentasikan data dan informasi dalam biner.
- Algoritma kriptografi modern beroperasi dalam mode bit (bandingkan dengan algoritma kriptografi klasik beroperasi dalam mode karakter)

  → kunci, plainteks, cipherteks, diproses dalam rangkaian bit

  → operasi **xor** paling banyak digunakan di dalam algoritmanya

- Meskipun disebut kriptografi modern, namun algoritmanya tetap menggunakan dua teknik dasar dasar di dalam kriptografi klasik: **teknik substitusi** dan **teknik transposisi (permutasi)**,

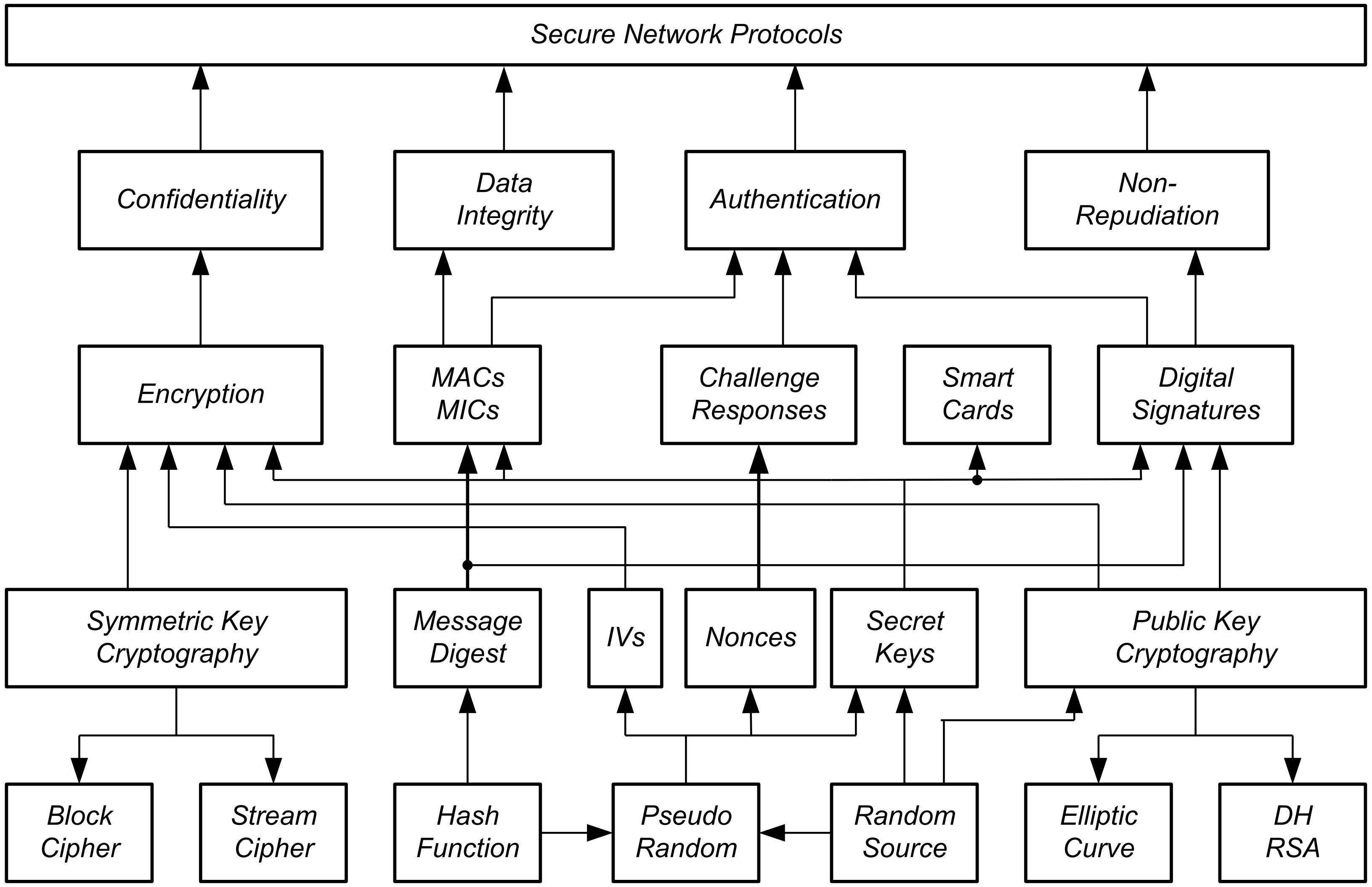
- tetapi operasinya dibuat lebih kompleks, tidak sesederhana cipher klasik.

  Tujuannya: agar *cipher* modern lebih sulit dikriptanalisis

- Selain kedua teknik dasar tersebut, juga digunakan teknik lain seperti rotasi, kompresi, ekspansi, penjumlahan modulo, dan lain-lain.

- Kriptografi modern melahirkan konsep-konsep baru seperti algoritma kriptografi kunci-publik, fungsi *hash*, protokol kriptografi, tanda-tangan digital, pembangkit bilangan acak, skema pembagian kunci, dsb.

Diagram Blok Kriptografi Modern



Bit, Byte, dan Kode Heksadesimal

- Pesan di dalam *cipher* modern dienkripsi bit-per-bit atau byte-per-byte, atau dalam kelompok bit (byte).

  1 byte = 8 bit

- Pada beberapa algoritma kriptografi, pesan direpresentasikan dalam kode heksadesimal (Hex).

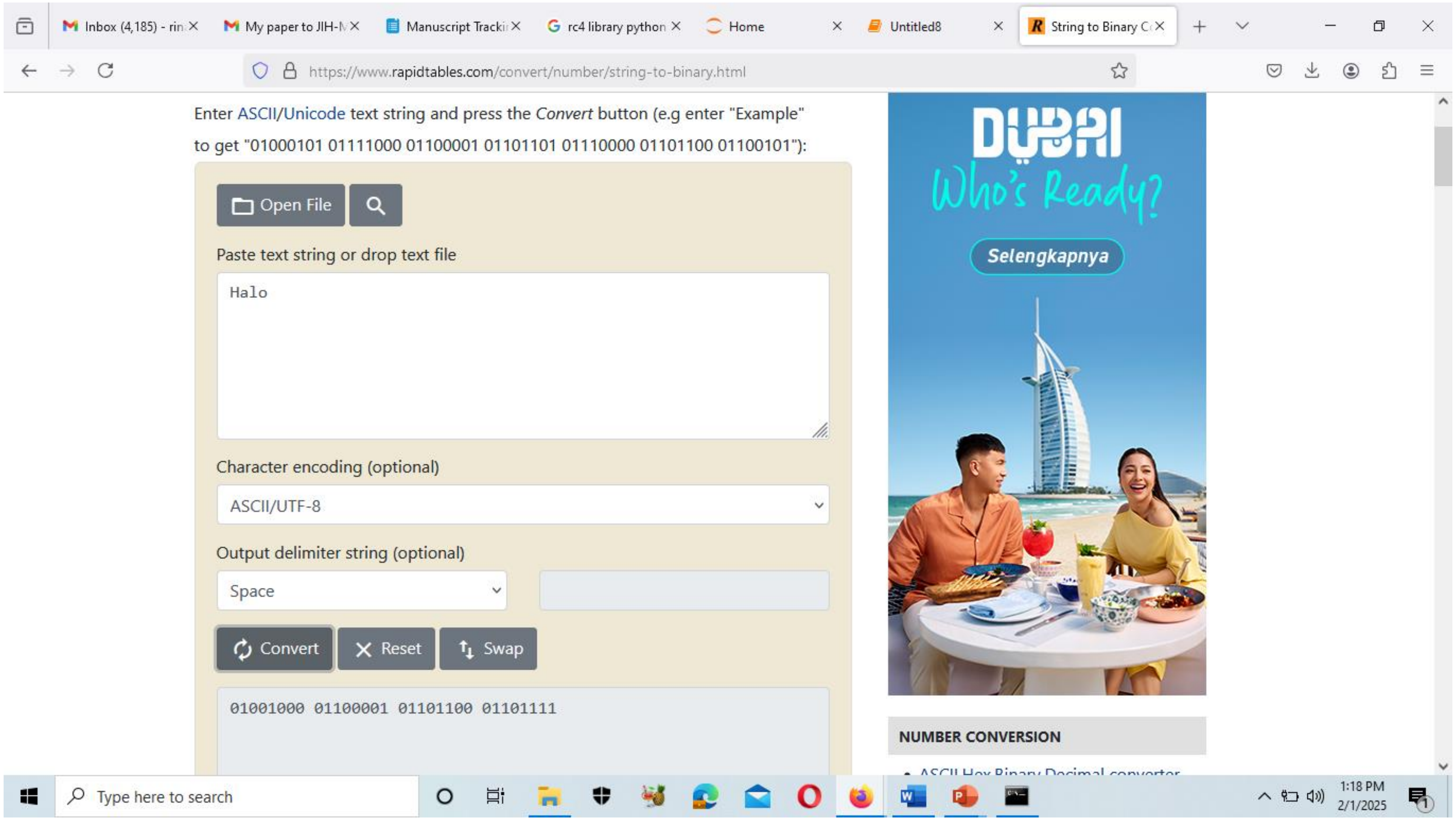
  1 kode hex = 4 bit

0000 = 0	0001 = 1	0010 = 2	0011 = 3
0100 = 4	0101 = 5	0110 = 6	0111 = 7
1000 = 8	1001 = 9	1010 = A	1011 = B
1100 = C	1101 = D	1110 = E	1111 = F

- Contoh: Pesan 100111010110 dalam kode Hex dengan cara membagi pesan menjadi blok 4-bit:

  1001 1101 0110 = 9D6

- Konversi teks ke biner: https://www.rapidtables.com/convert/number/string-to-binary.html



Enter ASCII/Unicode text string and press the *Convert* button (e.g enter "Example" to get "01000101 01111000 01100001 01101101 01110000 01101100 01100101"):

Paste text string or drop text file

```
Halo
```

Character encoding (optional): ASCII/UTF-8

Output delimiter string (optional): Space

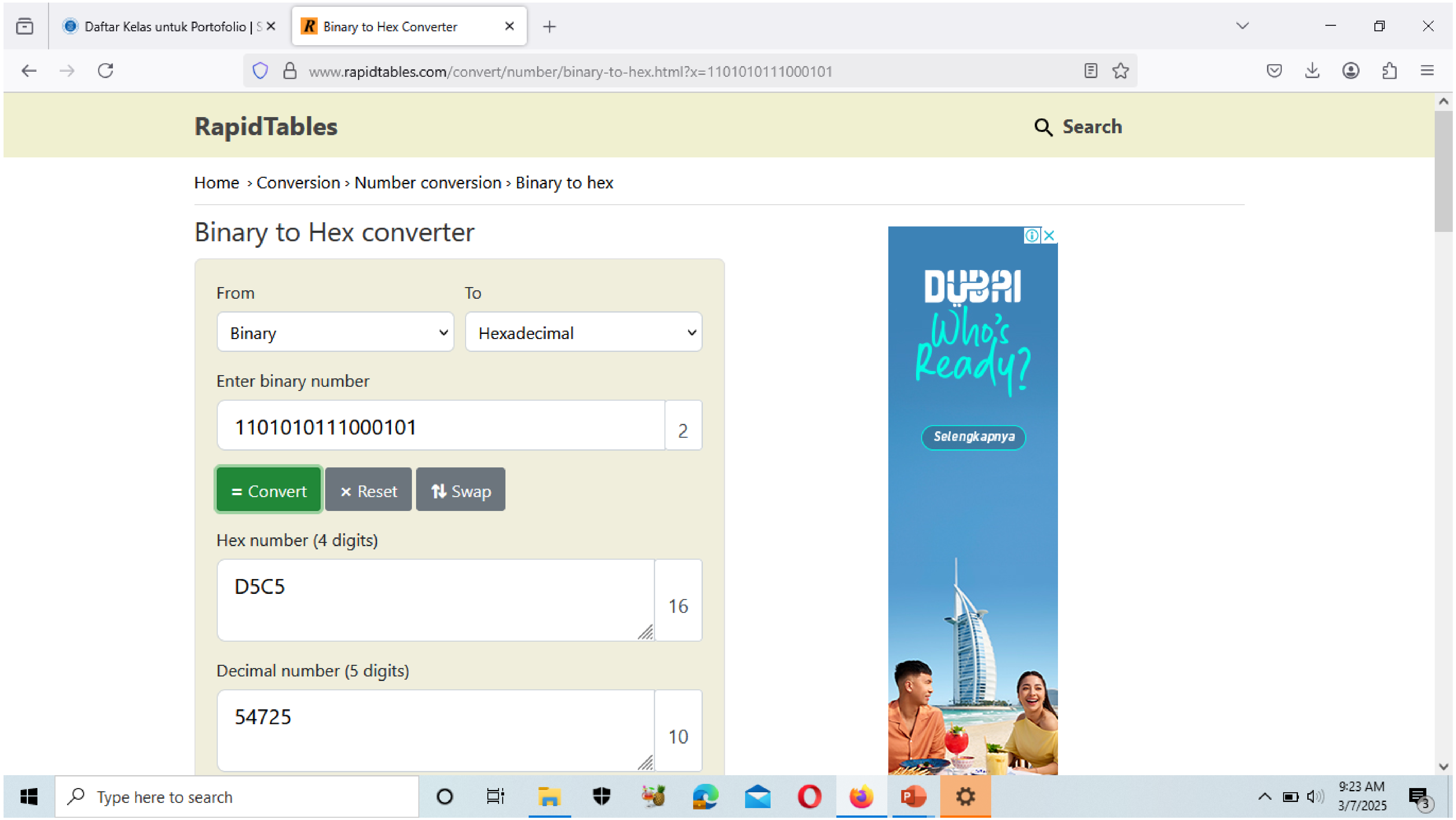
```
01001000 01100001 01101100 01101111
```

Contoh:

Halo → 01001000 01100001 01101100 01101111

Indonesia emas 2045 →01001001 01101110 01100100 01101111 01101110 01100101 01110011 01101001 01100001 00100000 01100101 01101101 01100001 01110011 00100000 00110010 00110000 00110100 00110101

Konversi biner ke hexa:



Daftar Kelas untuk Portofolio | S

Binary to Hex Converter

www.rapidtables.com/convert/number/binary-to-hex.html?x=1101010111000101

RapidTables

Search

Home › Conversion › Number conversion › Binary to hex

Binary to Hex converter

From: Binary

To: Hexadecimal

Enter binary number

1101010111000101 2

= Convert × Reset ⇅ Swap

Hex number (4 digits)

D5C5 16

Decimal number (5 digits)

54725 10

- Jika pesan diproses dalam kelompok bit, maka rangkaian bit pesan dibagi menjadi blok-blok bit berukuran sama.

- Contoh: Plainteks `100111010110001011100001`

  Bila dibagi menjadi blok 8-bit

  ```
  10011101   01100010 11100001
  ```

  atau dalam kode heksadesimal menjadi :

  ```
  9E   62   E1
  ```

- *Padding bits*: bit-bit tambahan jika ukuran blok terakhir tidak mencukupi panjang blok

- Contoh: Plainteks `100111010110`

  Bila dibagi menjadi blok 5-bit:

  `10011   10101` **000**`10`

Padding bits mengakibatkan ukuran cipherteks sedikit lebih besar daripada ukuran plainteks semula.

Operasi *XOR*

- Di dalam *cipher* alir maupun cipher blok, operasi XOR adalah operasi yang paling sering digunakan
- Notasi: $\oplus$
- Operasi:

$0 \oplus 0 = 0$ $\quad$ $0 \oplus 1 = 1$

$1 \oplus 0 = 1$ $\quad$ $1 \oplus 1 = 0$

a	b	$a \oplus b$
0	0	0
0	1	1
1	0	1
1	1	0

- Sifat-sifat operasi XOR:

  (i) $a \oplus a = 0$

  (ii) $a \oplus b = b \oplus a$

  (iii) $a \oplus (b \oplus c) = (a \oplus b) \oplus c$

Contoh:

(i) $1 \oplus 1 = 0$

(ii) $1 \oplus 0 = 0 \oplus 1 = 1$

(iii) $1 \oplus (0 \oplus 1) = (1 \oplus 0) \oplus 1 = 0$

Operasi *Bitwise* XOR

- Jika dua rangkaian dioperasikan dengan *XOR*, maka operasinya dilakukan dengan meng-*XOR*-kan setiap bit yang berkoresponden dari kedua rangkaian bit tersebut.

  Contoh: 10011 ⊕ 11001 = 01010
  yang dalam hal ini, hasilnya diperoleh sebagai berikut:

$$\begin{array}{cccccl} 1 & 0 & 0 & 1 & 1 & \\ 1 & 1 & 0 & 0 & 1 & \oplus \\ \hline 1 \oplus 1 & 0 \oplus 1 & 0 \oplus 0 & 1 \oplus 0 & 1 \oplus 1 & \\ 0 & 1 & 0 & 1 & 0 & \end{array}$$

Cipher Sederhana dengan operasi XOR

- Sama seperti *Vigenere Cipher*, tetapi dalam mode bit
- Setiap bit plainteks di-*XOR*-kan dengan setiap bit kunci.

Enkripsi: $C = P \oplus K$

Dekripsi: $P = C \oplus K$

Contoh:

plainteks	01100101		(karakter ‘e’)
kunci	00110101	$\oplus$	(karakter ‘5’)
cipherteks	01010000		(karakter ‘P’)
kunci	00110101	$\oplus$	(karakter ‘5’)
plainteks	01100101		(karakter ‘e’)

- Jika panjang bit-bit kunci lebih pendek daripada panjang bit-bit pesan, maka bit-bit kunci diulang penggunaannya secara periodik (seperti halnya pada Vigenere Cipher)

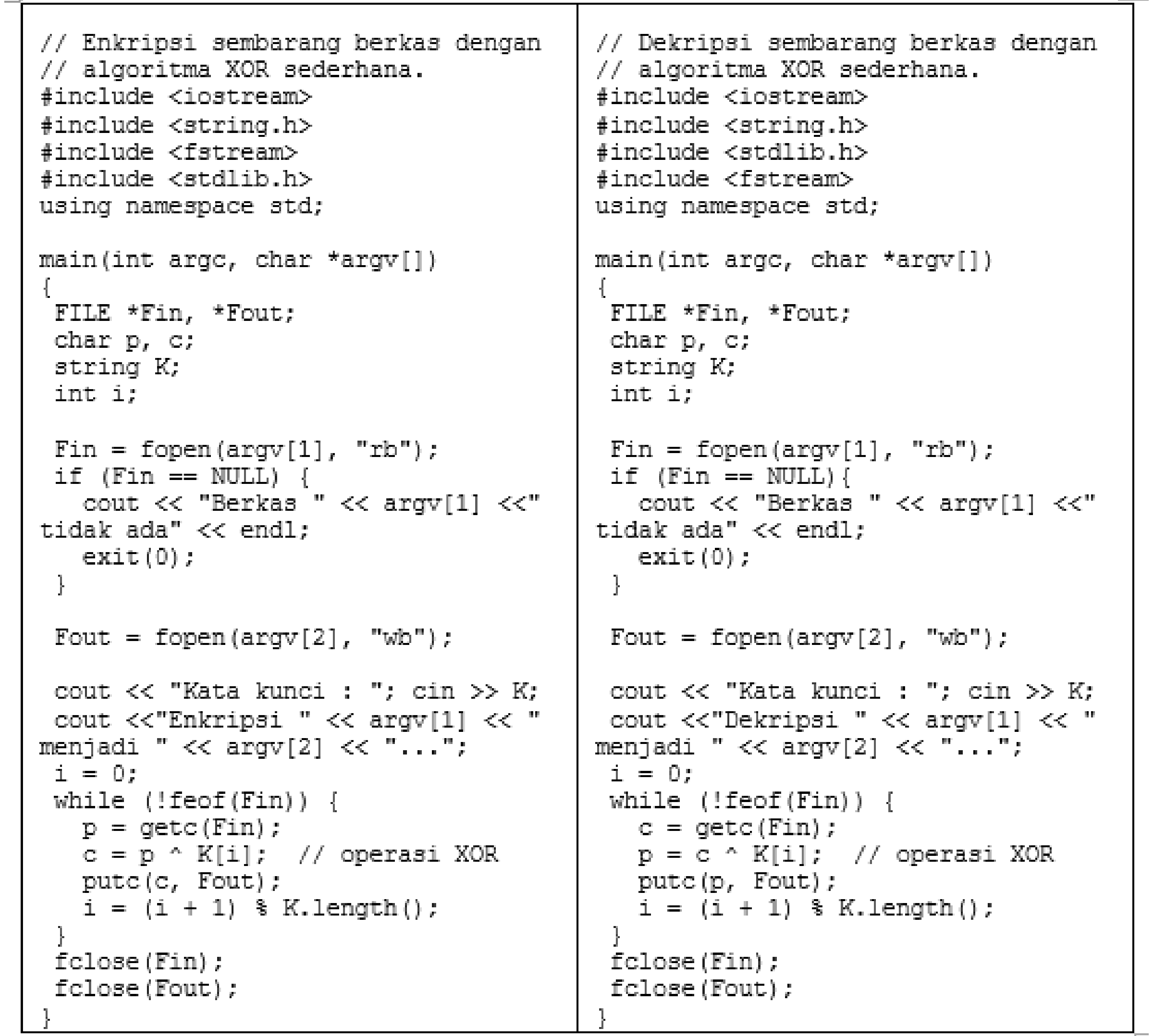
- Contoh:

  Plainteks : 10010010101110101010001110001

  Kunci : 11011011011011011011011011011

  Cipherteks: 01001001110101110001010101010

Program C++ untuk enkripsi-dekripsi file dengan cipher XOR sederhana

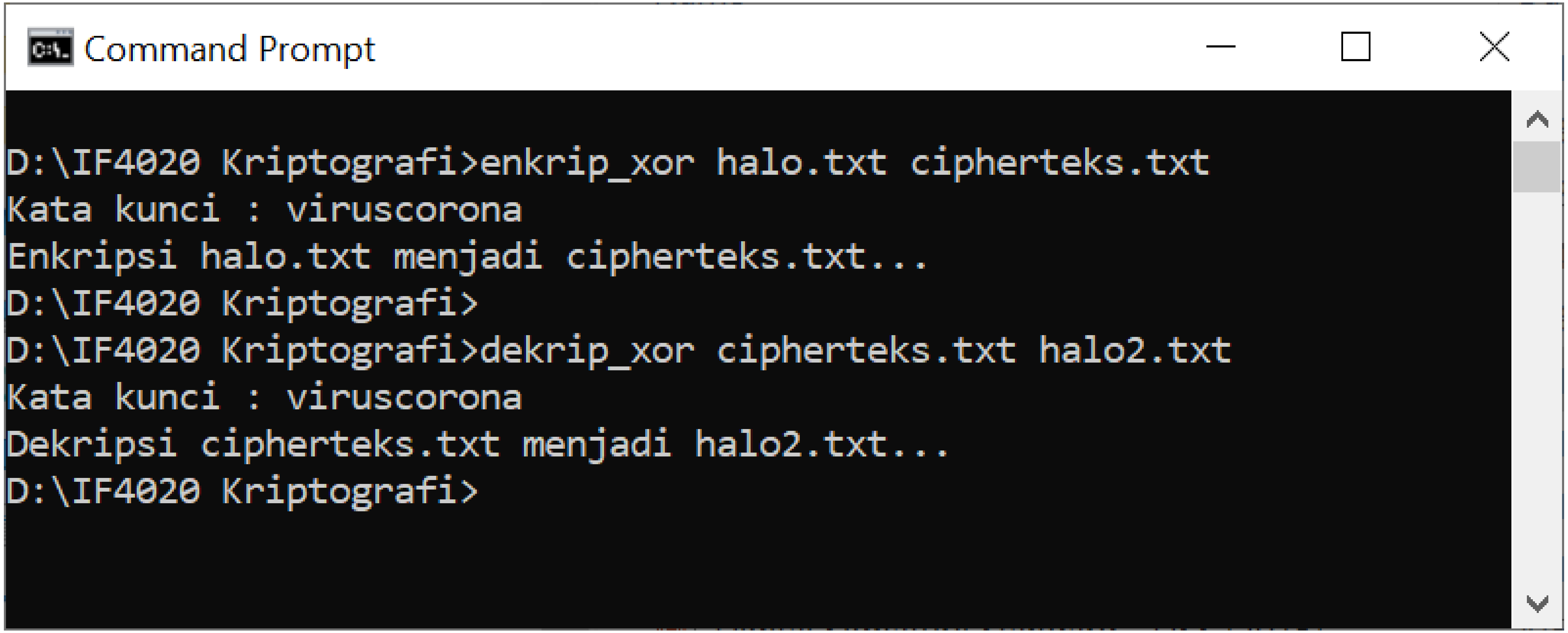


```
// Enkripsi sembarang berkas dengan
// algoritma XOR sederhana.
#include <iostream>
#include <string.h>
#include <fstream>
#include <stdlib.h>
using namespace std;

main(int argc, char *argv[])
{
 FILE *Fin, *Fout;
 char p, c;
 string K;
 int i;

 Fin = fopen(argv[1], "rb");
 if (Fin == NULL) {
   cout << "Berkas " << argv[1] <<"
tidak ada" << endl;
   exit(0);
 }

 Fout = fopen(argv[2], "wb");

 cout << "Kata kunci : "; cin >> K;
 cout <<"Enkripsi " << argv[1] << "
menjadi " << argv[2] << "...";
 i = 0;
 while (!feof(Fin)) {
   p = getc(Fin);
   c = p ^ K[i];  // operasi XOR
   putc(c, Fout);
   i = (i + 1) % K.length();
 }
 fclose(Fin);
 fclose(Fout);
}
```

(a) enkrip_xor.cpp

```
// Dekripsi sembarang berkas dengan
// algoritma XOR sederhana.
#include <iostream>
#include <string.h>
#include <stdlib.h>
#include <fstream>
using namespace std;

main(int argc, char *argv[])
{
 FILE *Fin, *Fout;
 char p, c;
 string K;
 int i;

 Fin = fopen(argv[1], "rb");
 if (Fin == NULL){
   cout << "Berkas " << argv[1] <<"
tidak ada" << endl;
   exit(0);
 }

 Fout = fopen(argv[2], "wb");

 cout << "Kata kunci : "; cin >> K;
 cout <<"Dekripsi " << argv[1] << "
menjadi " << argv[2] << "...";
 i = 0;
 while (!feof(Fin)) {
   c = getc(Fin);
   p = c ^ K[i];  // operasi XOR
   putc(p, Fout);
   i = (i + 1) % K.length();
 }
 fclose(Fin);
 fclose(Fout);
}
```

(b) dekrip_xor.cpp



Command Prompt

```
D:\IF4020 Kriptografi>enkrip_xor halo.txt cipherteks.txt
Kata kunci : viruscorona
Enkripsi halo.txt menjadi cipherteks.txt...
D:\IF4020 Kriptografi>
D:\IF4020 Kriptografi>dekrip_xor cipherteks.txt halo2.txt
Kata kunci : viruscorona
Dekripsi cipherteks.txt menjadi halo2.txt...
D:\IF4020 Kriptografi>
```

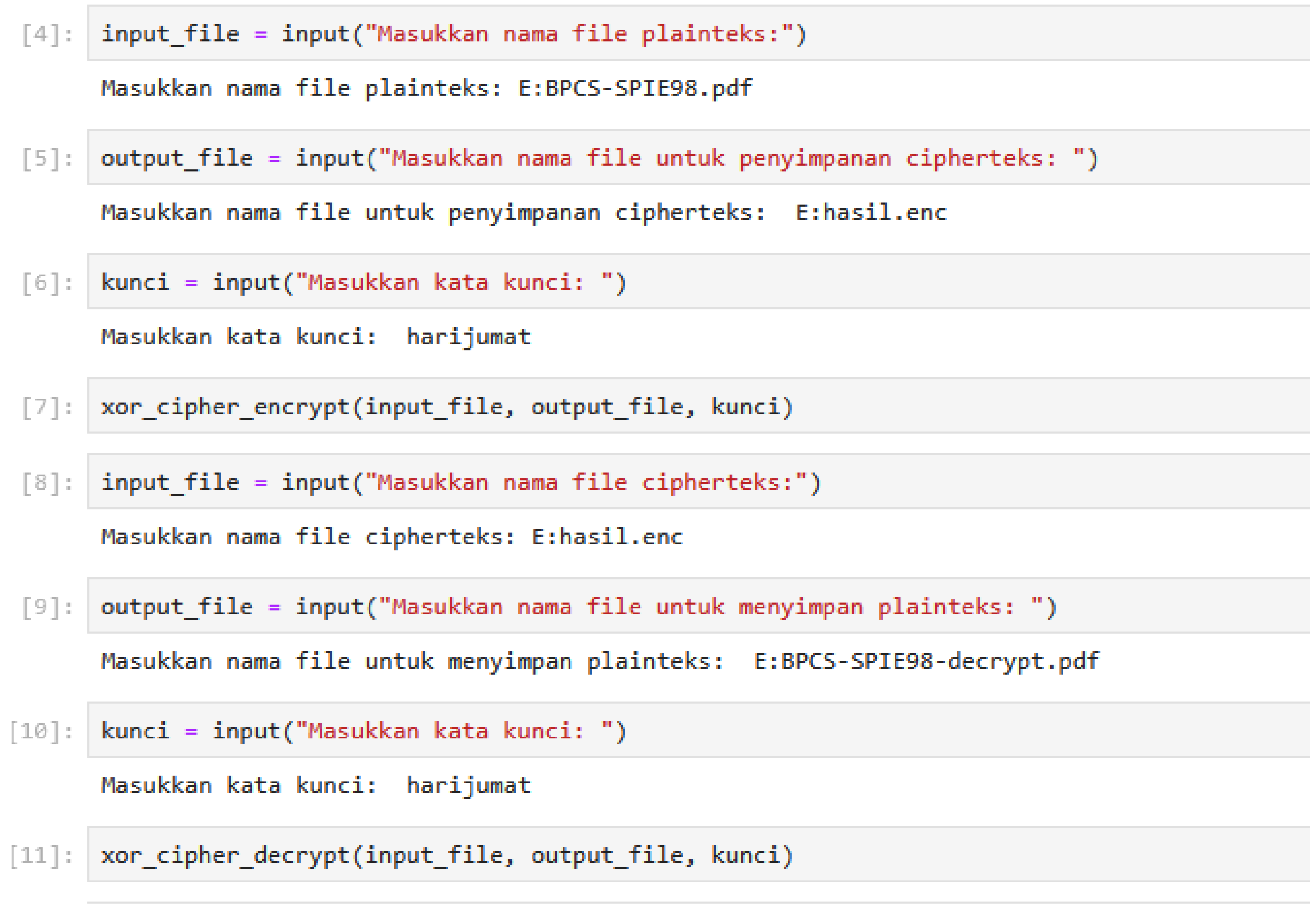
Program Python untuk enkripsi-dekripsi file dengan cipher XOR sederhana

```python
import sys
```

```python
def xor_cipher_encrypt(input_file, output_file, kunci):
    with open(input_file, 'rb') as file:
        plainteks = file.read()

    cipherteks = bytearray()
    byte_kunci = bytearray(kunci, 'utf-8')
    n = len(kunci)
    indeks_kunci = 0
    for byte in plainteks:
        c = byte ^ byte_kunci[indeks_kunci]
        cipherteks.append(c)
        indeks_kunci = (indeks_kunci + 1) % n

    with open(output_file, 'wb') as file:
        file.write(cipherteks)
```

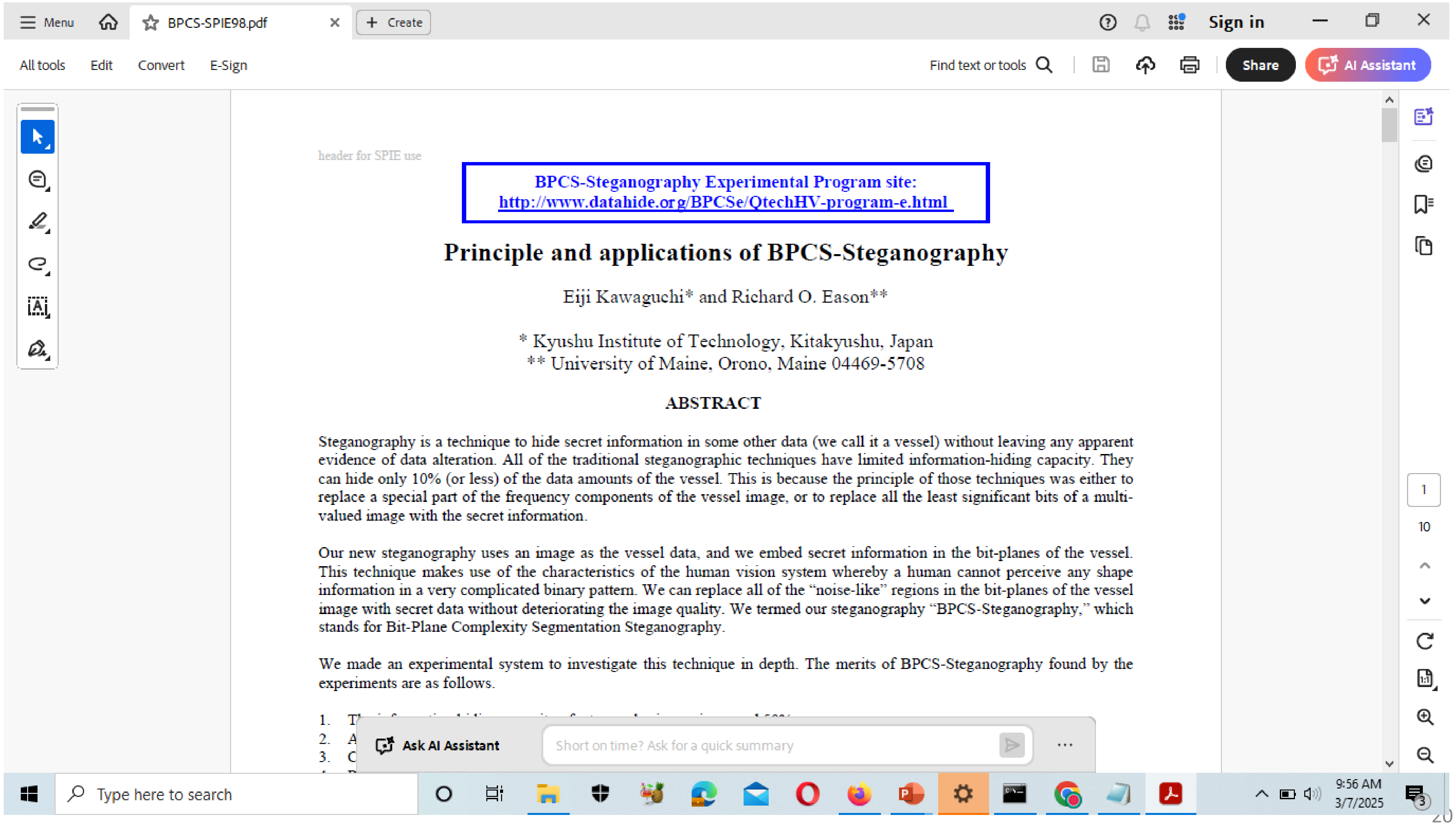
```python
def xor_cipher_decrypt(input_file, output_file, kunci):
    with open(input_file, 'rb') as file:
         cipherteks = file.read()

    plainteks = bytearray()
    byte_kunci = bytearray(kunci, 'utf-8')
    n = len(kunci)
    indeks_kunci = 0
    for byte in cipherteks:
        p = byte ^ byte_kunci[indeks_kunci]
        plainteks.append(p)
        indeks_kunci = (indeks_kunci + 1) % n

    with open(output_file, 'wb') as file:
        file.write(plainteks)
```

Contoh penggunaan:



```
[4]: input_file = input("Masukkan nama file plainteks:")

     Masukkan nama file plainteks: E:BPCS-SPIE98.pdf

[5]: output_file = input("Masukkan nama file untuk penyimpanan cipherteks: ")

     Masukkan nama file untuk penyimpanan cipherteks:  E:hasil.enc

[6]: kunci = input("Masukkan kata kunci: ")

     Masukkan kata kunci:  harijumat

[7]: xor_cipher_encrypt(input_file, output_file, kunci)

[8]: input_file = input("Masukkan nama file cipherteks:")

     Masukkan nama file cipherteks: E:hasil.enc

[9]: output_file = input("Masukkan nama file untuk menyimpan plainteks: ")

     Masukkan nama file untuk menyimpan plainteks:  E:BPCS-SPIE98-decrypt.pdf

[10]: kunci = input("Masukkan kata kunci: ")

     Masukkan kata kunci:  harijumat

[11]: xor_cipher_decrypt(input_file, output_file, kunci)
```

Contoh file plainteks: BPCS-SPIE98.pdf



header for SPIE use

BPCS-Steganography Experimental Program site:
http://www.datahide.org/BPCSe/QtechHV-program-e.html

Principle and applications of BPCS-Steganography

Eiji Kawaguchi* and Richard O. Eason**

* Kyushu Institute of Technology, Kitakyushu, Japan
** University of Maine, Orono, Maine 04469-5708

ABSTRACT

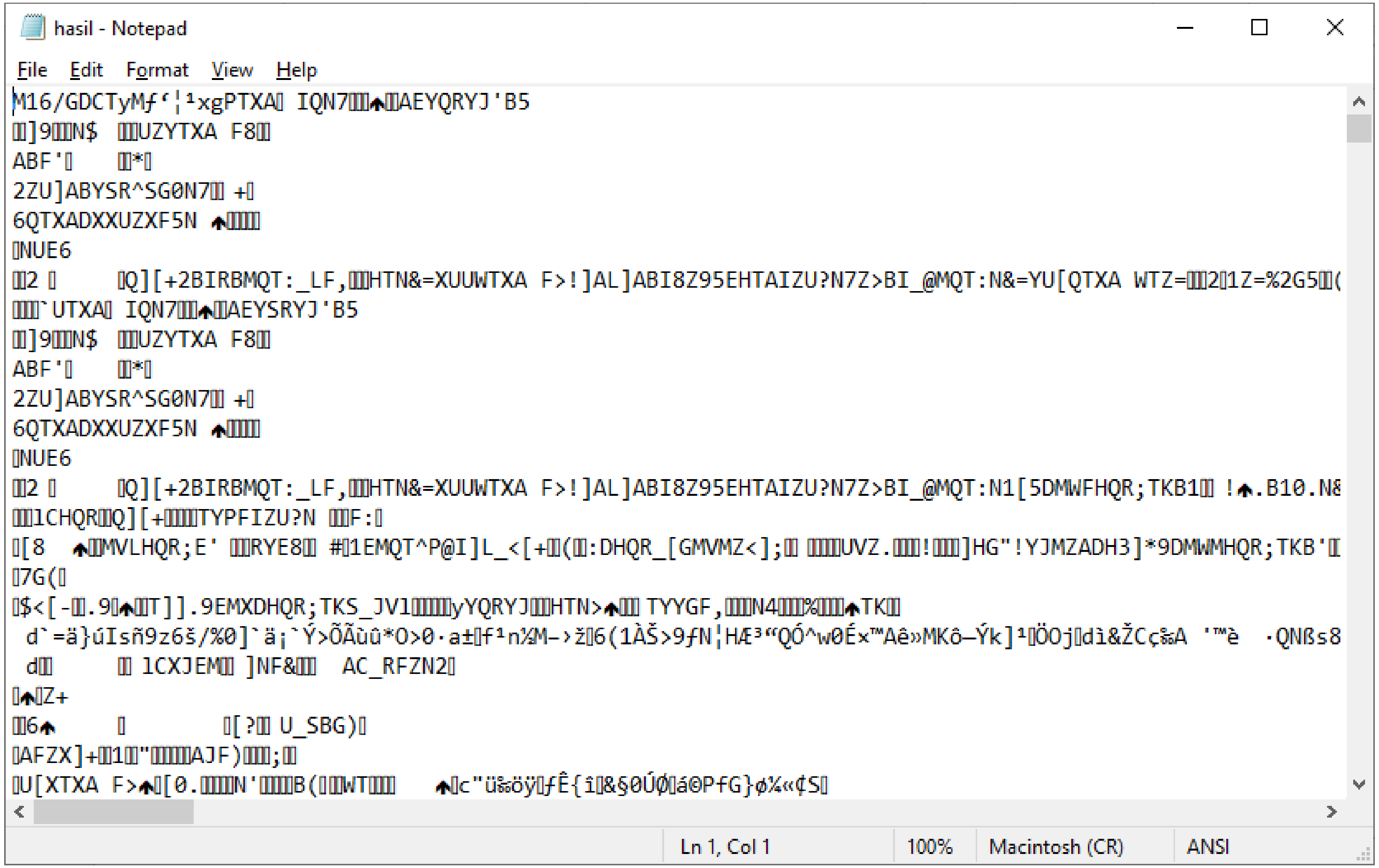
Steganography is a technique to hide secret information in some other data (we call it a vessel) without leaving any apparent evidence of data alteration. All of the traditional steganographic techniques have limited information-hiding capacity. They can hide only 10% (or less) of the data amounts of the vessel. This is because the principle of those techniques was either to replace a special part of the frequency components of the vessel image, or to replace all the least significant bits of a multi-valued image with the secret information.

Our new steganography uses an image as the vessel data, and we embed secret information in the bit-planes of the vessel. This technique makes use of the characteristics of the human vision system whereby a human cannot perceive any shape information in a very complicated binary pattern. We can replace all of the "noise-like" regions in the bit-planes of the vessel image with secret data without deteriorating the image quality. We termed our steganography "BPCS-Steganography," which stands for Bit-Plane Complexity Segmentation Steganography.

We made an experimental system to investigate this technique in depth. The merits of BPCS-Steganography found by the experiments are as follows.

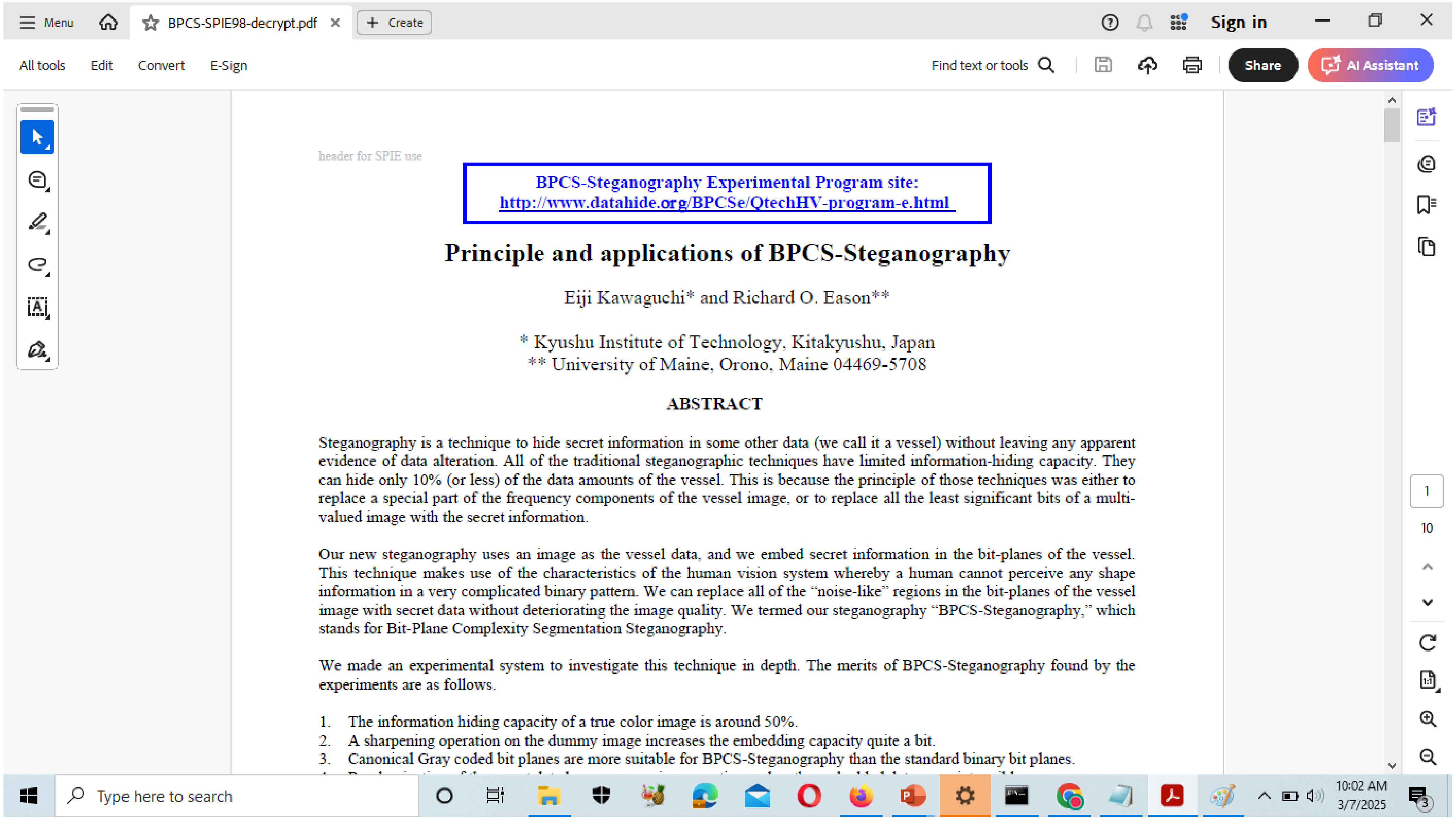
1. T
2. A
3. C

Contoh hasil file cipherteks: hasil.enc



```
M16/GDCTyMƒ‘¦¹xgPTXA▯ IQN7▯▯▲▯AEYQRYJ'B5
▯▯]9▯▯N$  ▯▯UZYTXA F8▯▯
ABF'▯    ▯▯*▯
2ZU]ABYSR^SG0N7▯▯ +▯
6QTXADXXUZXF5N ▲▯▯▯▯
▯NUE6
▯▯2 ▯     ▯Q][+2BIRBMQT:_LF,▯▯HTN&=XUUWTXA F>!]AL]ABI8Z95EHTAIZU?N7Z>BI_@MQT:N&=YU[QTXA WTZ=▯▯2▯1Z=%2G5▯▯(
▯▯▯`UTXA▯ IQN7▯▯▲▯AEYSRYJ'B5
▯▯]9▯▯N$  ▯▯UZYTXA F8▯▯
ABF'▯    ▯▯*▯
2ZU]ABYSR^SG0N7▯▯ +▯
6QTXADXXUZXF5N ▲▯▯▯▯
▯NUE6
▯▯2 ▯     ▯Q][+2BIRBMQT:_LF,▯▯HTN&=XUUWTXA F>!]AL]ABI8Z95EHTAIZU?N7Z>BI_@MQT:N1[5DMWFHQR;TKB1▯▯ !▲.B10.N&
▯▯1CHQR▯Q][+▯▯▯TYPFIZU?N ▯▯F:▯
▯[8  ▲▯▯MVLHQR;E' ▯▯RYE8▯▯ #▯1EMQT^P@I]L_<[+▯▯(▯▯:DHQR_[GMVMZ<];▯▯ ▯▯▯▯UVZ.▯▯▯!▯▯▯]HG"!YJMZADH3]*9DMWMHQR;TKB'▯▯
▯7G(▯
▯$<[-▯▯.9▯▲▯▯T]].9EMXDHQR;TKS_JV1▯▯▯▯yYQRYJ▯▯HTN>▲▯▯ TYYGF,▯▯▯N4▯▯▯%▯▯▯▲TK▯▯
 d`=ä}úIsñ9z6š/%0]`ä¡`Ý>ÕÃùû*O>0·a±▯f¹n½M–›ž▯6(1ÀŠ>9ƒN¦HÆ³“QÓ^w0É×™Aê»MKô–Ýk]¹▯ÖOj▯dì&ŽCç‰A '™è  ·QNßs8
 d▯▯     ▯▯ 1CXJEM▯▯ ]NF&▯▯  AC_RFZN2▯
▯▲▯Z+
▯▯6▲     ▯        ▯[?▯▯ U_SBG)▯
▯AFZX]+▯▯1▯▯"▯▯▯▯AJF)▯▯▯;▯▯
▯U[XTXA F>▲▯[0.▯▯▯▯N'▯▯▯▯B(▯▯▯WT▯▯▯   ▲▯c"ü‰öÿ▯ƒÊ{î▯&§0Ú▯▯á©PƒG}ø¼«¢S▯
```

Hasil dekripsi: BPCS-SPIE98-decrypt.pdf



header for SPIE use

BPCS-Steganography Experimental Program site:
http://www.datahide.org/BPCSe/QtechHV-program-e.html

Principle and applications of BPCS-Steganography

Eiji Kawaguchi* and Richard O. Eason**

* Kyushu Institute of Technology, Kitakyushu, Japan
** University of Maine, Orono, Maine 04469-5708

ABSTRACT

Steganography is a technique to hide secret information in some other data (we call it a vessel) without leaving any apparent evidence of data alteration. All of the traditional steganographic techniques have limited information-hiding capacity. They can hide only 10% (or less) of the data amounts of the vessel. This is because the principle of those techniques was either to replace a special part of the frequency components of the vessel image, or to replace all the least significant bits of a multi-valued image with the secret information.

Our new steganography uses an image as the vessel data, and we embed secret information in the bit-planes of the vessel. This technique makes use of the characteristics of the human vision system whereby a human cannot perceive any shape information in a very complicated binary pattern. We can replace all of the "noise-like" regions in the bit-planes of the vessel image with secret data without deteriorating the image quality. We termed our steganography "BPCS-Steganography," which stands for Bit-Plane Complexity Segmentation Steganography.

We made an experimental system to investigate this technique in depth. The merits of BPCS-Steganography found by the experiments are as follows.

1. The information hiding capacity of a true color image is around 50%.
2. A sharpening operation on the dummy image increases the embedding capacity quite a bit.
3. Canonical Gray coded bit planes are more suitable for BPCS-Steganography than the standard binary bit planes.

- Cipher XOR sederhana tidak aman, karena mudah dikriptanalisis dengan metode yang sama seperti metode Kasiski

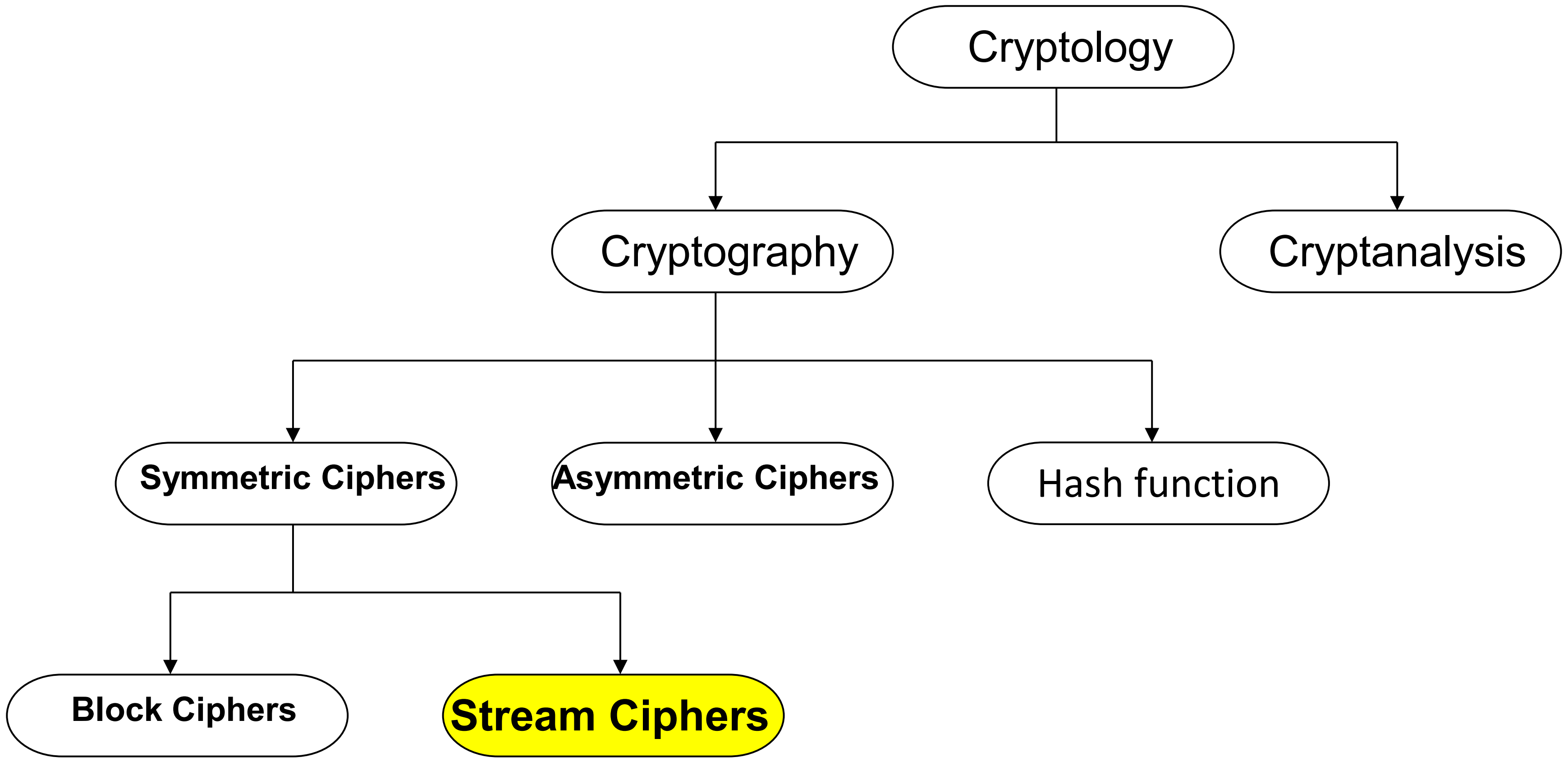
Kategori *cipher* berbasis bit

1. *Cipher* Alir (*Stream Cipher*)

- beroperasi pada bit secara individual
- enkripsi/dekripsi pesan secara bit per bit dengan operasi XOR

2. *Cipher* Blok (*Block Cipher*)

- beroperasi pada blok-blok bit (sekumpulan bit)
  (contoh: 128-bit/blok = 16 karakter/blok)
- enkripsi/dekripsi pesan secara blok per blok bit


Cryptology
Cryptography
Cryptanalysis
Symmetric Ciphers
Asymmetric Ciphers
Hash function
Block Ciphers
Stream Ciphers