Fundamentals of Cryptography

an interactive tutorial

Eindhoven

Henk van Tilborg
Eindhoven University of Technology

2.1 Classical Systems

2.1.1 Caesar Cipher

Shift each letter in the text cyclicly over k places. So, with $k = 7$ one gets the following encryption of the word cleopatra (note that the letter z is mapped to a):

cleopatra $\xrightarrow{+1}$ dmfpqbusb $\xrightarrow{+1}$ engqrcvtc $\xrightarrow{+1}$ fohrsdwud $\xrightarrow{+1}$ gpistexve $\xrightarrow{+1}$ hqjtufywf $\xrightarrow{+1}$ irkuvgzxg $\xrightarrow{+1}$ jslvwhayh

To do this in *Mathematica*, we need modular arithmetic (replace a by 0, b by 1,…,z by 25 and make your calculations modulo 26).

```
CaesarCipher[plaintext_, key_] :=     FromCharacterCode[
  Mod[ ToCharacterCode[plaintext] - 97 + key, 26] + 97]
```

```
plaintext = "cleopatrawasanegyptianqueeen";
key = 7;
CaesarCipher[plaintext, key]
```

```
jslvwhayhdhzhulnfwaphuxblllu
```

An easy way to break the system is to try out all possible keys. This method is called exhaustive key search.

The cryptanalysis of the ciphertext "xyuysuyifvyxi".

```
ciphertext = "xyuysuyifvyxi";
Table[{key, CaesarCipher[ciphertext, -key]},
  {key, 0, 10}] // TableForm
```

```
0    xyuysuyifvyxi
1    wxtxrtxheuxwh
2    vwswqswgdtwvg
3    uvrvprvfcsvuf
4    tuquoquebrute
5    stptnptdaqtsd
6    rsosmosczpsrc
7    qrnrlnrbyorqb
8    pqmqkmqaxnqpa
9    oplpjlpzwmpoz
10   nokoikoyvlony
```

So, the key k was $-4 \equiv 22 \pmod{26}$.

2.1.2 Vigenère Cryptosystem

The *Vigenère cryptosystem* (1586) consists of r Caesar ciphers applied periodically. In the example below, the key is a word of length $r = 7$. The i-th letter in the key defines the particular Caesar cipher that is used for the encryption of the letters i, $i + r$, $i + 2r$, … in the plaintext.

EXAMPLE

We identify {0, 1, …, 25} with {*a*, *b*, …, *z*}. With the key "michael" one gets the following encipherment:

plaintext	a	c	r	y	p	t	o	s	y	s	t	e	m	o	f	t	e	n	i	s	a	c	o	m	p	r	o	m
key	m	i	c	h	a	e	l	m	i	c	h	a	e	l	m	i	c	h	a	e	l	m	i	c	h	a	e	l
ciphertext	m	k	t	f	p	x	z	e	g	u	a	e	q	z	r	b	g	u	i	w	l	o	w	o	w	r	s	x

Vigenère used the following table to realize this:

0	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
1	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a
2	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b
3	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c
4	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d
5	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e
6	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f
7	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g
8	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h
9	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i
10	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j
11	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k
12	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l
13	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m
14	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n
15	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
16	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
17	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q
18	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r
19	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s
20	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t
21	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u
22	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v
23	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w
24	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x
25	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y

The Vigenère Table

In *Mathematica*, addition of two letters can be realized in a similar way, as our earlier implementation of the Caesar cipher:

```
AddTwoLetters[a_, b_] := FromCharacterCode[
  Mod[(ToCharacterCode[a] - 97) +
     (ToCharacterCode[b] - 97), 26] + 97]
```

By means of the *Mathematica* functions `StringTake` and `StringLength`, and the function AddTwoLetters, defined above, encryption with the Vigenère cryptosystem can be realized as follows:

```
plaintext = "typehereyourplaintextinsmallletters";
key = "keyword";
ciphertext = "";
Do[ciphertext =
   ciphertext <> AddTwoLetters[StringTake[plaintext, {i}],
     StringTake[key, {Mod[i - 1, StringLength[key]] + 1}]],
  {i, 1, StringLength[plaintext]}];
ciphertext
```

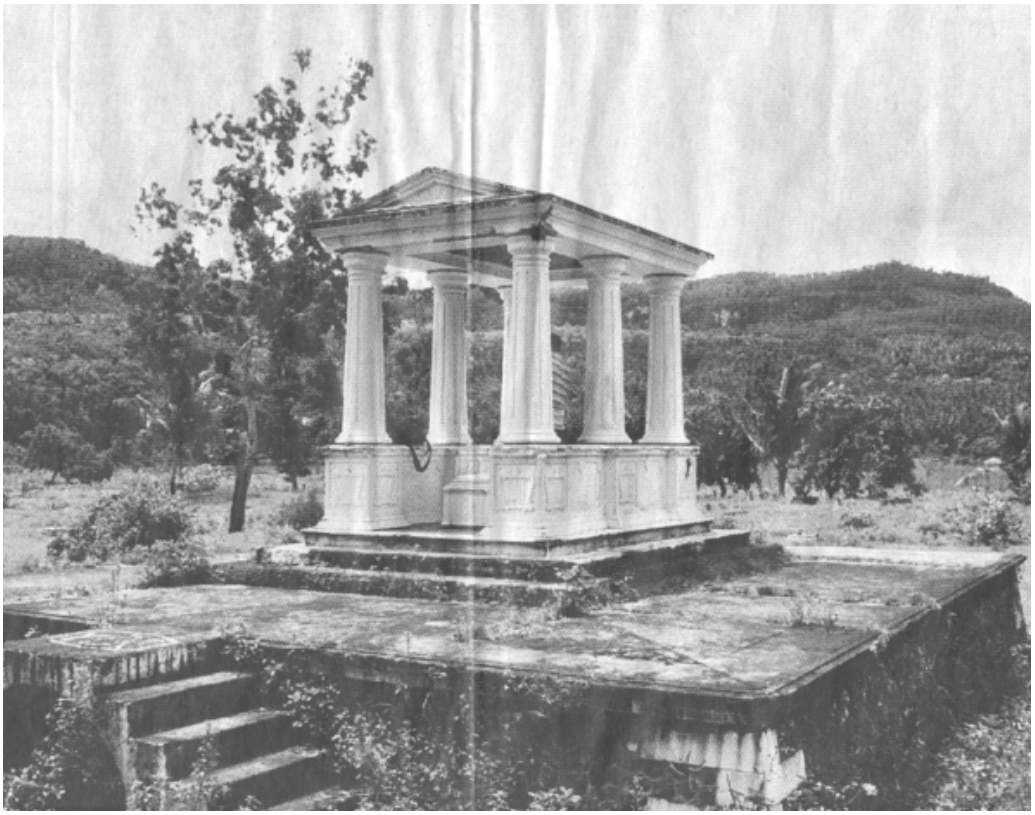
```
dcnavvuocmqfgokmlpsowsrqiocovirpsiv
```

For centuries, no one had an effective way of breaking this system, mainly because one did not have a technique of determining the key length *r*. Once one knows *r*, one can find the *r* simple substitutions by grouping together the letters *i*, $i + r$, $i + 2r$, …, for each *i*, $0 \le i < r$, and break each of these *r* simple substitutions individually.

In 1863, the Prussian army officer, F.W. Kasiski, solved the problem of finding the key length *r* by statistical means. We shall demonstrate this method by means of an example.

EXAMPLE

On a gravestone close to the city of Patjitan on the island Java in Indonesia (Rudy Kousbroek, NRC, March 29, 2002),



Het Mysterie van Patjitan

Een graftombe aan de Javaanse zuidkust draagt een lange inscriptie in code. Generaties lang zijn alle pogingen haar te ontcijferen vergeefs geweest.

one can find the following text.


```
ciphertext = "z_gjxfxfmf
    _dfwefdxwbujhrvwwgzebgzzzbagkbmagxbzmqhwgenfwb
    _vwohwxkywobml_dnnxokwwbmvxksotxkxwbmvxzdeubqfxddfm-
     fdfalmvxocwgbdfhjxfwgzebgzzfvmjhlrntkrsklmzhblgxqh-
     cnjxfxihwyadwgengzxbzmfmyubsmbddfwbgwxidoxoddwfrer-
     azskqnwmbjdxfmrtihcnlnam-
     t_vxoyaxkzdlbqwxkkwobmalehwkkzetxkk
    ____zvsztkrage_____swvrrrkfqpxzjm__gzldvxkvwkayg
    ___lorhdzbfghfcvt__gffjrticwgjnwb__cxkvwzluwkdndzls-
     ztkdexkdfntdwkqdjnduagadfmlsoxadjsfdfl";
L = StringLength[ciphertext]
```

```
403
```

From

```
i = 4;
Do[If[
  Count[Characters[ StringTake[ ciphertext, {m, m + i - 1}]] -
     Characters[ StringTake[ ciphertext, {n, n + i - 1}]],
    0] == i, Print[{m, n, StringTake[ ciphertext,
     {m, m + i - 1}], FactorInteger[n - m]}]],
 {m, 1, L - i}, {n, m + 1, L - i + 1} ]
```

```
{4, 160, jxfx, {{2, 2}, {3, 1}, {13, 1}}}
{27, 127, wgze, {{2, 2}, {5, 2}}}
{28, 128, gzeb, {{2, 2}, {5, 2}}}
{29, 129, zebg, {{2, 2}, {5, 2}}}
{30, 130, ebgz, {{2, 2}, {5, 2}}}
{31, 131, bgzz, {{2, 2}, {5, 2}}}
{44, 176, xbzm, {{2, 2}, {3, 1}, {11, 1}}}
{50, 170, wgen, {{2, 3}, {3, 1}, {5, 1}}}
{66, 250, wobm, {{2, 3}, {23, 1}}}
{79, 91, wbmv, {{2, 2}, {3, 1}}}
{80, 92, bmvx, {{2, 2}, {3, 1}}}
{267, 281, ____, {{2, 1}, {7, 1}}}
{267, 282, ____, {{3, 1}, {5, 1}}}
{273, 365, sztk, {{2, 2}, {23, 1}}}
{281, 282, ____, {{1, 1}}}
{307, 351, xkvw, {{2, 2}, {11, 1}}}
```

we conclude that, almost surely, the key length is 4.

Note that we even found twice the same word of length 8:

```
i = 8
Do[If[
  Count[Characters[ StringTake[ ciphertext, {m, m + i - 1}]] -
     Characters[ StringTake[ ciphertext, {n, n + i - 1}]],
    0] == i, Print[{m, n, StringTake[ ciphertext,
     {m, m + i - 1}], FactorInteger[n - m]}]],
 {m, 1, L - i}, {n, m + 1, L - i + 1} ]
```

```
8
```

```
{27, 127, wgzebgzz, {{2, 2}, {5, 2}}}
```

We now analyse the subsequences of coordinates that are equal to 0, 1, 2 reps. 3 modulo 4.

```
r = 4;
subtext = Table[{}, {i, 1, r}];
Do[
  subtext[[Mod[i - 1, r] + 1]] = subtext[[Mod[i - 1, r] + 1]] <>
    StringTake[ciphertext, {i}], {i, 1, L}];
Do[Print[subtext[[i]]], {i, 1, r}]
```

```
zxmfduvzzambhn_hymnkmskmdqddmcdxzzmrrmlhxhdnbmsdgddrznjmhn
  _yzqkmhzk_sr__rfz_dvy_hfc_jcn_vunsdddudsdd

_ffwxjwezgazwfvwwlnwvoxvefffvwffezjnszgcfwwgzymfwodeswdrcavadww-
   awek_za_srqjgvwgldgvgrwwcwwdzefwjafojf

gx_ewhwbzkgmgwwxo
  _xwxtwxuxmaxghwbfhtkhxnxygzmubwxxwrkmxtnmxxlxolkt_ztg_wrpmzxk
  _ozhtftgbxzkztxnkngmxsl

jfdfbrggbbxqebokbdobkxbzbdflobjggvlklbqjiaexfbdbiofaqbfiltokbkb-
   ekx_vke_vkx_lka_rbf_fij_kldlkktqdalaf
```

For the coordinates that are 0 modulo 4, we get the following statistics.

```
Table[{FromCharacterCode[j],
  Count[ToCharacterCode[subtext[[1]]], j]}, {j, 97, 122}]
```

```
{{a, 1}, {b, 2}, {c, 3}, {d, 17}, {e, 0},
 {f, 3}, {g, 1}, {h, 7}, {i, 0}, {j, 2}, {k, 4}, {l, 1},
 {m, 11}, {n, 7}, {o, 0}, {p, 0}, {q, 2}, {r, 5}, {s, 5},
 {t, 0}, {u, 3}, {v, 3}, {w, 0}, {x, 3}, {y, 3}, {z, 9}}
```

Assuming a Caesar cipher, we conclude that very likely $a \to z$ has been used.

Similarly,

```
Table[{FromCharacterCode[j],
  Count[ToCharacterCode[subtext[[2]]], j]}, {j, 97, 122}]
```

```
{{a, 6}, {b, 0}, {c, 3}, {d, 5}, {e, 6}, {f, 13},
 {g, 7}, {h, 0}, {i, 0}, {j, 5}, {k, 1}, {l, 2}, {m, 1},
 {n, 2}, {o, 3}, {p, 0}, {q, 1}, {r, 3}, {s, 3}, {t, 0},
 {u, 0}, {v, 7}, {w, 20}, {x, 2}, {y, 1}, {z, 7}}
```

Very likely we have $a \to s$ on the coordinates 1 mod 4.

```
Table[{FromCharacterCode[j],
  Count[ToCharacterCode[subtext[[3]]], j]}, {j, 97, 122}]
```

```
{{a, 1}, {b, 4}, {c, 0}, {d, 0}, {e, 1}, {f, 2},
 {g, 8}, {h, 5}, {i, 0}, {j, 0}, {k, 7}, {l, 3}, {m, 7},
 {n, 4}, {o, 3}, {p, 1}, {q, 0}, {r, 2}, {s, 1}, {t, 8},
 {u, 2}, {v, 0}, {w, 10}, {x, 19}, {y, 1}, {z, 7}}
```

Quite likely we have $a \to t$ on the coordinates 2 mod 4.

```
Table[{FromCharacterCode[j],
  Count[ToCharacterCode[subtext[[4]]], j]}, {j, 97, 122}]
```

```
{{a, 5}, {b, 16}, {c, 0}, {d, 6}, {e, 4}, {f, 9},
 {g, 4}, {h, 0}, {i, 4}, {j, 4}, {k, 12}, {l, 8},
 {m, 0}, {n, 0}, {o, 5}, {p, 0}, {q, 4}, {r, 2}, {s, 0},
 {t, 2}, {u, 0}, {v, 3}, {w, 0}, {x, 5}, {y, 0}, {z, 1}}
```

And $a \to x$ on the coordinates 3 mod 4.

```
SubTwoLetters[a_, b_] :=
 FromCharacterCode[Mod[(ToCharacterCode[a] - 97) -
     (ToCharacterCode[b] - 97), 26] + 97]
```

We guess the key "zstx".

```
key = "zstx";
plaintext = "";
Do[plaintext = If[StringTake[ciphertext, {i}] == "_",
    plaintext = plaintext <> "_",
    plaintext = plaintext <> SubTwoLetters[
       StringTake[ciphertext, {i}],     StringTake[
        key, {Mod[i - 1, StringLength[key]] + 1}]]],
 {i, 1, StringLength[ciphertext]}]
plaintext
```

```
a_nmyneinn_ggeliefdevrouwedjamijahgeboreninachttienhonde
  _ddrieenzevent_goverledendentwaalfdendecembernegentienhonder-
   deneenomyndjamijahmynroosvansaronhoemoetikumyneliefdeenhoog-
   achtingbetuigendeheelewereldismydaartoetekleinzalikuooit-
   w_derzienalsereenlevenishiernamaals
  ____gythansinh_____adyszyngywaart__ogoedenwerdzo
  ___tvuilgegooidda__omikzaldenmoei
  __kenwegovergolgothanemenenuweerterugvindentotwederziens
```

□ Reconstruction

From this we can reconstruct:

> **Aan myne innig geliefde vrouwe Djamijah, geboren in achttienhonderd drie en zeventig overleden den twaalfden December negentienhonderd en een. O myn Djamijah, myn roos van Saron, hoe moet ik u myne liefde en hoogachting betuigen? De heele wereld is my daar toe te klein. Zal ik u ooit wederzien? Als er een leven is hiernamaals zoudt gy thans in het paradys zyn. Gy waart zoo goed en werd zoo met vuil gegooid daarom. Ik zal den moeilyken weg over Golgotha nemen en u weer terugvinden. Tot wederziens!**

where the red letters are our guesses.

Does the combination "zstx" originate from the Dutch word "pijn"?

```
ciphertext = "zstx";
Table[{key, CaesarCipher[ciphertext, -key]},
  {key, 0, 10}] // TableForm
```

```
0     zstx
1     yrsw
2     xqrv
3     wpqu
4     vopt
5     unos
6     tmnr
7     slmq
8     rklp
9     qjko
10    pijn
```

Her husband was Marcus Jacobus van Erp Taalman-Kip who married her when he was 71!

2.2 Block Ciphers

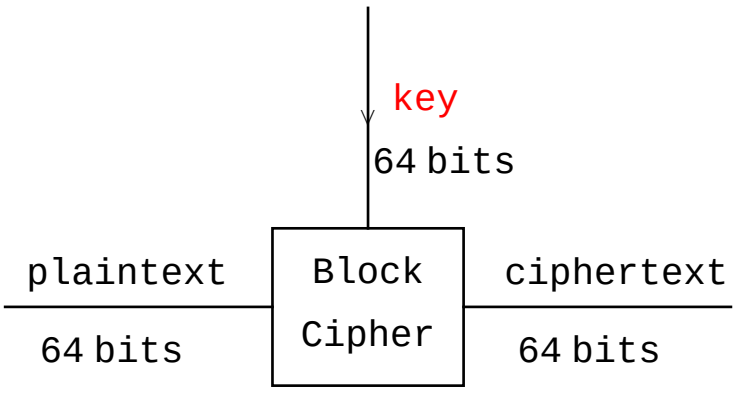
2.2.1 Some General Principles

Block ciphers handle *n* bits at a time.

They have no memory (to store previous input).

There can operate at very high speeds.

- **Electronic Codebook**

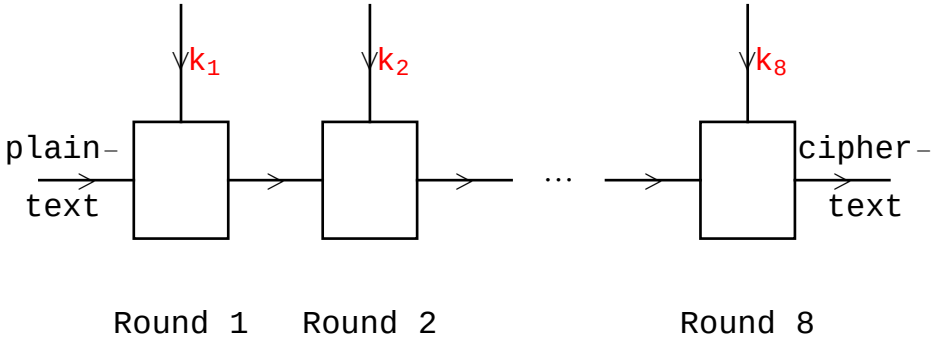


Often, the same device can be used for encryption and decryption.

Typically, the block cipher consists of a sequence of identical looking rounds each operating under a *round key* that is computed from the key *k*.

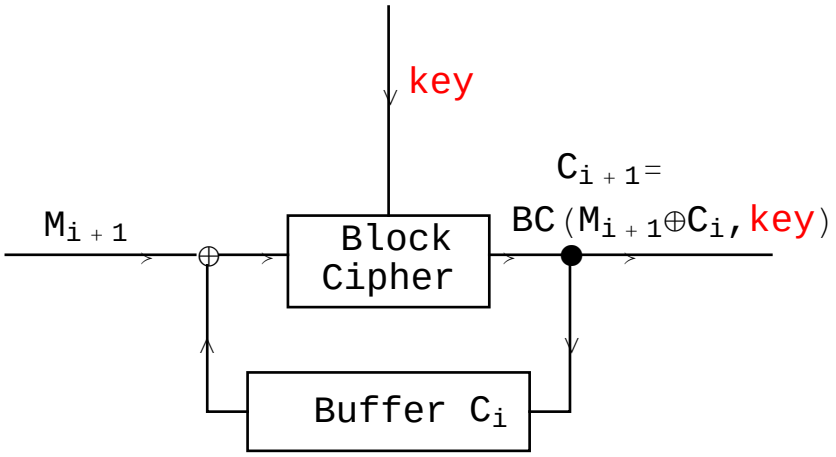
Each round is designed to realize "confusion" and "diffusion" in order to obscure

dependencies and other statistical properties of the plaintext.

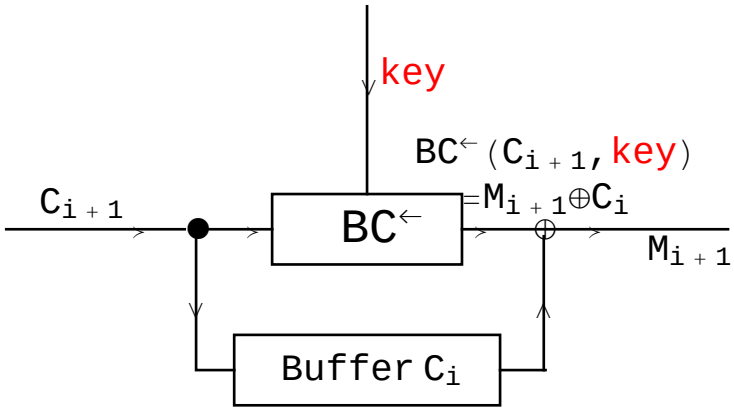


Note that the same plaintext will result in the same ciphertext as long as the key has not been changed. To avoid this situation feedback is introduced. Examples are given below.

□ Cipher-block Chaining and Message Authentication Codes



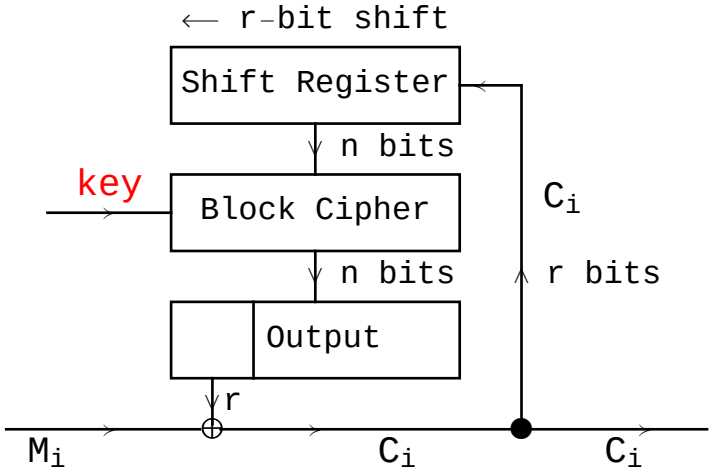
The buffer needs to be initialized with a value that also the reciever needs to know.



This form of chaining can also be used for message authentication: send the plaintext M_0, M_1, …, M_n followed by the last ciphertext C_n as Message Authentication Code (MAC).

□ Cipher Feedback

Ideal when you often have to encrypt short messages, say *r* bits at a time (think of telnet session).



□ An Identity Verification Protocol

A smart card and a card reader want to verify each others authenticity.

On the smart card of Bob is stored

- **an identity number Id_{Bob}**
- **a secret key k_{Bob}.**

The secret key k_{Bob} has been put there by the card issuer and is the result of an encryption of the identity number Id_{Bob} under a master key MK.

$$k_{Bob} = BC(MK, Id_{Bob}).$$

The smart card presents Id_{Bob} to the card reader.

The master key MK is "stored" in the card reader.

A genuine card reader can compute k_{Bob} from

$$k_{Bob} = BC(MK, Id_{Bob}).$$

How does the card reader check the authenticity of the smart card?

The card reader presents a random string r of n bits to the smart card.

If the smart card can present $BC(k_{Bob}, r)$ to the card reader, it must know k_{Bob}.

How does the smart card check the authenticity of the card reader?

The smart card presents a random string r of n bits to the card reader.

If the card reader can present $BC(k_{Bob}, r)$ to the card, it must be able to compute k_{Bob} from Id_{Bob}, which means that it must know MK.

2.2.2 DES

2.2.3 Triple DES

2.2.4 IDEA

2.2.5 Advanced Encryption Standard (AES)

A collection of proposals has been studied by the (American) National Institute of Standards and Technology (NIST for short) for a new industrial standard for a block cipher.

The names of these proposals are CAST-256, CRYPTON, DFC, DEAL, E2, FROG, HPC, LOKI97, MAGENTA, MARS, RC6, RIJNDAAEL, SAFER+, SERPENT and TWOFISH (see the web page Advanced Encryption Standard).

The second round of the selection was concluded in August 1999. The following contenders remained in the race: MARS (IBM), RC6TM (RSA Laboratories), RIJNDAEL (Daemen and Rijmen), SERPENT (Ross Anderson, Eli Biham, and Lars Knudsen) and TWOFISH (Bruce Schneier e.a.).

On October 2, 2000, the final selection was made: RIJNDAEL!

2.2.6 Rijndael

Like most modern block ciphers, Rijndael is an iterated block cipher: it specifies a transformation, also called the round function, and the number of times this function is iterated on the data block to be encrypted/decrypted, also referred to as the number of

rounds.

Rijndael is a substitution-linear transformation network, i.e. it is not a Feistel-type block cipher like e.g. DES. The block length and the key length can be independently specified to 128, 192, and 256 bits.

The number of rounds in Rijndael depends in the following way on the the block length and the key length.

cipher\key	128	192	256
128	10	12	14
192	12	12	14
256	14	14	14

The round function consists of the following operations:

- **ByteSub (affects individual bytes),**
- **ShiftRow (shifts rows),**
- **MixColumn (affects each column),**
- **RoundKey addition (overall XOR).**

These are applied to the intermediate cipher result, also called the State: a 4×4, 4×6, resp. 4×8 matrix of which the entries consist of 8 bits, i.e. one byte. For example, when the block length is 192, one gets

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$

where each $a_{i,j}$ consists of 8 bits, so it has the form $\{(a_{i,j})_0, (a_{i,j})_1, \ldots, (a_{i,j})_7\}$. For example, $a_{0,0} = \{1, 0, 1, 1, 0, 0, 0, 1\}$.

Sometimes, we use the one-dimensional ordering (columnwise) i.e.
$a_{0,0}, a_{1,0}, a_{2,0}, a_{3,0}, a_{0,1}, \ldots, a_{3,5}$.

We define N_b as the number of columns in the array above. So, the the block cipher length is 32 N_b bits, or 4 N_b bytes (each byte consists of 8 bits), or N_b 4-byte words.

Similarly, the Cipher Key length consists of 32 N_k bits, or 4 N_k bytes, or N_k 4-byte words.

□ One Round

ByteSub

This is the only non-linear part in each round.

Apply to each byte $a_{i,j}$ two operations:

1) Interpret $a_{i,j}$ as element in $\mathrm{GF}(2^8)$ and replace it by its multiplicative inverse,

if it is not 0, otherwise leave it the same.

2) Replace the resulting 8-tuple, say $(x_0, x_1, \ldots, x_7)$ by

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}.$$

The finite field GF(2^8) is made by means of the irreducible polynomial $m(\alpha) = 1 + \alpha + \alpha^3 + \alpha^4 + \alpha^8$. This polynomial is not primitive!

Note that both operations are invertible.

```
<<FiniteFields`
```

```
f256 = GF[2, {1, 1, 0, 1, 1, 0, 0, 0, 1}];
one = f256[{1, 0, 0, 0, 0, 0, 0, 0}]
α = f256[{0, 1, 0, 0, 0, 0, 0, 0}]
```

```
{1, 0, 0, 0, 0, 0, 0, 0}_2
```

```
{0, 1, 0, 0, 0, 0, 0, 0}_2
```

```
in = {0, 1, 0, 0, 0, 0, 0, 0};
pol = Sum[in[[i]] α^(i-1), {i, 1, 8}]
inver = 1 / pol
```

```
{0, 1, 0, 0, 0, 0, 0, 0}_2
```

```
{1, 0, 1, 1, 0, 0, 0, 1}_2
```

```
A = {{1, 0, 0, 0, 1, 1, 1, 1},
     {1, 1, 0, 0, 0, 1, 1, 1},
     {1, 1, 1, 0, 0, 0, 1, 1},
     {1, 1, 1, 1, 0, 0, 0, 1},
     {1, 1, 1, 1, 1, 0, 0, 0},
     {0, 1, 1, 1, 1, 1, 0, 0},
     {0, 0, 1, 1, 1, 1, 1, 0},
     {0, 0, 0, 1, 1, 1, 1, 1}};

b = {1, 1, 0, 0, 0, 1, 1, 0};
Mod[A.inver[[1]] + b, 2]
```

```
{1, 1, 1, 0, 1, 1, 1, 0}
```

Instead of performing these calculations, one can also replace them by one substitution table: the ByteSub S-box.

ShiftRow

The rows of the State are shifted cyclically to the left using different offsets: do not shift row 0, shift row 1 over c_1 bytes, row 2 over c_2 bytes, and row 3 over c_3 bytes, where

	c_1	c_2	c_3
128	1	2	3
192	1	2	3
256	1	3	4

.

So

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$

becomes

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$
$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$	$a_{1,0}$
$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$	$a_{2,0}$	$a_{2,1}$
$a_{3,3}$	$a_{3,4}$	$a_{3,5}$	$a_{3,0}$	$a_{3,1}$	$a_{3,2}$

MixColumn

Interpret each column as a polynomial of degree 3 over GF(2^8) and multiply it with

$(1+\alpha)\,x^3+x^2+x+\alpha$

modulo x^4+1.

Note that the above polynomial is invertible modulo x^4+1.

```
g[x_] = (1 + α) x^3 + one x^2 + one x + α
```

```
{0, 1, 0, 0, 0, 0, 0, 0}2 + x {1, 0, 0, 0, 0, 0, 0, 0}2 +
 x^2 {1, 0, 0, 0, 0, 0, 0, 0}2 + x^3 {1, 1, 0, 0, 0, 0, 0, 0}2
```

Suppose that the first column looks like

```
col = {1 + α + α^3 + α^6 + α^7, one, α^2 + α^4 + α^5 + α^6, α};
col // TableForm
```

```
{1, 1, 0, 1, 0, 0, 1, 1}2
{1, 0, 0, 0, 0, 0, 0, 0}2
{0, 0, 1, 0, 1, 1, 1, 0}2
{0, 1, 0, 0, 0, 0, 0, 0}2
```

```
colpol[x_] = col[[1]] + col[[2]] x + col[[3]] x^2 + col[[4]] x^3
```

```
x^2 {0, 0, 1, 0, 1, 1, 1, 0}2 + x^3 {0, 1, 0, 0, 0, 0, 0, 0}2 +
 x {1, 0, 0, 0, 0, 0, 0, 0}2 + {1, 1, 0, 1, 0, 0, 1, 1}2
```

```
ownexpand[expr_] :=
 Collect[expr /. {GF → GF$}, x] /. {GF$ → GF}
```

```
pr[x_] = ownexpand[colpol[x] * g[x]]
prod[x_] = PolynomialMod[pr[x], x^4 - 1]
```

```
x^2 {0, 1, 0, 0, 0, 1, 0, 0}2 +
 x^6 {0, 1, 1, 0, 0, 0, 0, 0}2 + x^5 {0, 1, 1, 1, 1, 0, 0, 1}2 +
 x {1, 0, 0, 1, 0, 0, 1, 1}2 + x^4 {1, 0, 1, 0, 1, 1, 1, 0}2 +
 {1, 0, 1, 1, 0, 0, 0, 1}2 + x^3 {1, 1, 1, 0, 1, 1, 0, 0}2
```

```
{0, 0, 0, 1, 1, 1, 1, 1}₂ + x² {0, 0, 1, 0, 0, 1, 0, 0}₂ +
 x {1, 1, 1, 0, 1, 0, 1, 0}₂ + x³ {1, 1, 1, 0, 1, 1, 0, 0}₂
```

The inverse operation is a multiplication by

```
h[x_] = (1 + α + α³) x³ + (1 + α² + α³) x² + (1 + α³) x + (α + α² + α³) ;
ownexpand[PolynomialMod[g[x] * h[x], x⁴ - 1]]
```

```
{1, 0, 0, 0, 0, 0, 0, 0}₂
```

```
ownexpand[PolynomialMod[prod[x] * h[x], x⁴ - 1]]
```

```
x² {0, 0, 1, 0, 1, 1, 1, 0}₂ + x³ {0, 1, 0, 0, 0, 0, 0, 0}₂ +
 x {1, 0, 0, 0, 0, 0, 0, 0}₂ + {1, 1, 0, 1, 0, 0, 1, 1}₂
```

Round Key Addition

XOR the whole matrix with a similar sized matrix (i.e. the Round Key) obtained from the cipher key in a way that depends on the round index.

Note that the XOR applied to a byte, really is an XOR applied to the 8 bits in the byte.

For example, if

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$

$\oplus$

$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$	$k_{0,4}$	$k_{0,5}$
$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$	$k_{1,4}$	$k_{1,5}$
$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$	$k_{2,4}$	$k_{2,5}$
$k_{3,0}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$	$k_{3,4}$	$k_{3,5}$

$=$

$u_{0,0}$	$u_{0,1}$	$u_{0,2}$	$u_{0,3}$	$u_{0,4}$	$u_{0,5}$
$u_{1,0}$	$u_{1,1}$	$u_{1,2}$	$u_{1,3}$	$u_{1,4}$	$u_{1,5}$
$u_{2,0}$	$u_{2,1}$	$u_{2,2}$	$u_{2,3}$	$u_{2,4}$	$u_{2,5}$
$u_{3,0}$	$u_{3,1}$	$u_{3,2}$	$u_{3,3}$	$u_{3,4}$	$u_{3,5}$

.

with $u_{0,0} = a_{0,0} \oplus k_{0,0}$, the coordinate-wise exclusive or.

```
a_{0,0} = {1, 1, 1, 1, 0, 0, 0, 0}; k_{0,0} = {1, 1, 0, 0, 1, 0, 1, 0};
Mod[a_{0,0} + k_{0,0}, 2]
```

```
{0, 0, 1, 1, 1, 0, 1, 0}
```

There is also an initial Round Key addition and one final round that differs slightly from the others (the MixColumn is omitted) .