DFG(V)X

- Cipher
- Description
- Background
- Security

○ ADFGX ◉ ADFGVX

Plaintext:

```
this is Arjun vankani information security practical eleven
```

Ciphertext (final result after the transposition step):

```
DF VA GG FX VG FA FG AA XX FD VG GG XX VF VX VG VX GA XF VA AD FA VV XX DV VD VA VF GF DV GF
AX DV AV FX GF FV VA DF DA FV AD XV DV XX VD GG VX VV XD AX FX
```

○ Unblocked ◉ Blocks of 2 ○ Blocks of 5

☑ Intermediate result after the substitution step (this is the first of two steps):

```
GV XG VA DD VA DD AF VV GX DG XX FF AF XX XV AF XX VA VA XX FD XD VV VD AF GV VA XD XX DD VG
FA DG VV VA GV DF GF VV VG FA GV VA FA AF XF VG XF VG FF VG XX
```

○ Unblocked ◉ Blocks of 2 ○ Blocks of 5

Options — keys and transposition key alphabet (length: 36 characters)

Substitution matrix

&	A	D	F	G	V	X
A	6	Q	A	1	0	G
D	X	S	Y	U	W	D
F	C	F	V	2	5	8
G	4	Z	P	3	T	J

V	I	M	7	E	R	B
X	9	O	L	H	K	N

Substitution key (to arrange substitution matrix):

Specify matrix by yourself: 6QA10GXSYUWDCFV2584ZP3TJIM7ERB9OLHKN

Random matrix

Standard matrix

Select route for substitution key: Rows from top left

Transposition key (entered by human): INFORMATIONSECURITY

Transposition key (thus used after rearranging: as string and as sorted number sequence):

5-9-4-11-13-8-1-16-6-12-10-15-3-2-18-14-7-17-19 (ACEFIIIMNNOORRSTTUY)

Parse transposition key alphabet | Close substitution options

CTOAUTHORS: Christian Sieche, Phil Pilcrow, David Kuche

The encoding procedure according to ADFGVX consists of two phases. For the first phase (substitution), the Polybius cipher is used. A matrix with 6 rows and columns is formed. Each character from the alphabet A-Z has to be written down in this matrix as well as the numbers 0-9. The predecessor substitution ADFGX used a matrix with only 5 rows and columns. In addition to that, the encoding procedure according to ADFGX is analogous to ADFGVX, which will be explained below.

Such a matrix could look like this:

	A	**D**	**F**	**G**	**V**	**X**
A	J	E	5	C	L	1
D	D	3	4	7	A	2
F	X	N	S	0	U	P
G	M	F	K	Z	8	9
V	I	6	Q	V	W	B
X	T	G	Y	O	R	H

For the message "GEHEIMNACHRICHT" the character G will be substituted according to the matrix shown above to XD. (row X and column D). The E is replaced by AD (row A and column D) and the H by XX (row X and column X). After substituting all characters we get: "XD AD XX AD VA

[redacted] FD DV AG XX XV VA AG XX XA".
[redacted]nsposition is used for the second phase of encoding. The matrix is read in row by row and read [redacted]t column by column. Also, an arbitrary keyword has to be chosen. Let's assume the key would be [redacted]YKEY".

	Y	K	E	Y
X	D	A	D	X
X	A	D	V	A
G	A	F	D	D
V	A	G	X	X
X	V	V	A	A
G	X	X	X	A

For the last step, the columns are swapped. This happens by sorting the keyword alphabetically. The result would be this matrix:

E	K	M	Y	Y
D	A	X	D	X
V	D	X	A	A
D	F	G	A	D
X	G	V	A	X
A	V	X	V	A
X	X	G	X	A

The character M of the keyword has been header of the first column before swapping.
Now it has become column 3.
The character Y of the keyword has been header of the second column before swapping.
Now it has become column 4.
The character K of the keyword has been header of the third column before swapping.
Now it has become column 2.
In result the original columns have been swapped to the positions (3-4-2-1-5).

To get the ciphertext, the matrix has to be read out column by column from the top to the bottom. The final ciphertext would be: "DV DX AX AD FG VX XX GV XG DA AA VX XA DX AA".

ADFGX and the successor ADFGVX were developed by the German intelligence officer Fritz Nebel (* 1891; † 1967). ADFGX was used for the first time at the 5th of March in 1918 during World War I. Only a few months later, on the 1st of June, an extended version of this cipher called ADFGVX was used. A group of German cryptologists considered this cipher to be unbreakable. Since the German troops were close to Paris, it was crucial for the allied forces to break this cipher and get intelligence about the next movements of the German troops. On the 2nd of June, the French crypt analyst Geoges Painvin managed to break the encoding for a German radio message. A little later he managed to decrypt a message which revealed the position of the German troops. This was arguably one of the most important reasons why the German attack failed.[2] The transmitted messages of the Germans were only composed of the characters A, D, F, G, V and X. These characters have been chosen because they are easily distinguishable in the Morse alphabet.[2]

(1) Singh, Simon: „Geheime Botschaften", Carl Hanser Verlag, 1999, p. 132
(2) Singh, Simon: „Geheime Botschaften", Carl Hanser Verlag, 1999, p. 448

To decode an ADFGVX cipher, the structure of the substitution matrix has to be known as well as the key for the transposition. However, the result of the substitution step is only a monoalphabetic substitution of the characters, which is not very secure. The transposition procedure is mainly responsible for the security of the cipher and an attacker has to find out how this procedure works.

e first phase of encoding results in a monoalphabetic substitution. Without the second phase s cipher would not be more secure than the Caesar cipher.

tokey

- Cipher
- Description
- Background
- Security
- About alphabets

Plaintext:
```
this is prectial of
autokey cipher
```

Encrypted text:
```
tyrmv lwzwk lxrpq
yiuec pesvw zlcti
```

Key: arjun

Options:
- [ ] filter whitespace characters
- [x] group 5 characters
- [x] filter non-alphabet characters
- [ ] convert to first alphabet
- [x] filter key on alphabet characters

Alphabets:

ABCDEFGHIJKLMNOPQRSTU\

abcdefghijklmnopqrstu\

Add alphabet

Details for the Message Alphabets

(chars)

×

Compressed alphabet notation:

Offset:

Keyword for alphabet:

Reverse | Permutate | Rotate left | Rotate right | Transform to other case | Add A–Z | Add a–z | Add 0–9 | Delete alphabet

The Autokey cipher is based on the Vigenère cipher but avoids the problem of periodically repeating a keyword. If the keyword is shorter than the plaintext, the plaintext is simply added to the keyword.

Plaintext: THIS IS A SECRET TEXT
Keyword: KEY
Key: KEYTHISISASECRETT
Ciphertext: DLGL PA S AWCJIV KIQM

The Autokey cipher was developed by Blaise de Vigenère as well. It is primarily based on the same methods as the Vigenère cipher, but it includes a modification, which increases the security of the cipher. This modification is based on ideas from Gerolamo Cardano.

The Autokey cipher is more secure than the Vigenère cipher, because a pattern search with the Kasiski- or the Friedman-Test leads to no result with the Autokey cipher. On the other hand, this cipher is not very secure when the attacker knows some parts of the plaintext because the plaintext is part of the key. Also, characters can be identified with analytical methods. Because parts of the keyword are derived from the plaintext, one can assume that the keyword contains natural language. If one would further assume that English has been used for the plaintext, the most frequently occurring character in the plaintext as well as in the key should be E. In a sufficiently long encoded text, the most frequently occurring character would then be the I which is the result of mapping the character E according to the key E. With the same procedure, every other character in the message can be determined. [1]

(1) Lang, H.W.: „Klassische Kryptografie“, http://www.iti.fh-flensburg.de/lang/krypto/klassisch.htm, Date: 2009-02-20

What is an alphabet?

An alphabet[1] is an ordered set of all characters which can occur in a plaintext, a secret text, or the key. "Ordered" means that sorting is possible and we can speak of the *n*-th character of an alphabet.

For classical methods, the alphabet often consists only of the uppercase letters (A-Z). Characters not belonging to the alphabet are not encrypted or allowed as keys.

The handling of non-alphabet characters (convert, skip, ...) can be set in the options - but this is not a function of the actual encryption process itself. This requires additional meta-information of the letters that must be recorded before encryption. Also, there is no general match on how to handle digits or special characters. Instead of performing a transformation before encryption, this implementation allows several alphabets to be specified (see below), thereby accomplishing the same within the encryption process.

Our implementation of **Vigenère**, **Beaufort**, etc. does not work internally with letters, but with numbers. Therefore, a translation must take place, which can on the one hand transform letters in numbers and, conversely, re-generate letters again. A function that performs this is called an alphabet function.

Let *s* be such a reversible function. Then the **Vigenère** encryption for an input character *in* and a key *key* can be described as:

$out = s^{-1}(s(in) + s(key))$

The letters of *in* and *key* are converted into numbers, these numbers are added, and the sum is re-converted to a letter. The conversion to letters takes place modulo to the alphabet length: If a 1 is added to the last character, the result of the sum is the first character of the alphabet.

How to describe an alphabet?

phabets (yes, there may be several: more below) can be described by a list *L* of letters. The habet function s_L returns the smallest index at which it occurs to a letter that is present in *L*. The lex of the first character can be configured. For letters that do not occur in *L*, the alphabet nction s_L is undefined.

Although the function is well-defined when a letter occurs more than once, this makes little sense in encryption algorithms, since the reversibility suffers. A corresponding warning is displayed.

The inverse function returns the *n*-th character for a number *n* in *L*. To *n*, the length of the list *L* is added or subtracted as often as necessary until the index lies in the list.

Example of an alphabet

For example, take the list *L* = "ABCD", whose length is 4. The message "ACDC" should be encrypted with the key "ABBA" according to the **Vigenère** method. The following steps take place:

in	$s_L(in)$	*key*	$s_L(key)$	$s_L(in) + s_L(key)$	$s_L^{-1}(s_L(in) + s_L(key))$
A	0	A	0	0	A
C	2	B	1	3	D
D	3	B	1	4	A
C	2	A	0	2	C

In the example, an overflow has occurred in the third letter, so that modulo $|L| = 4$ is calculated.

Shortcuts for alphabets

In order to simplify the representation of the alphabets, the following abbreviation has been introduced: The minus sign in the following letter 1-letter 2 is extended to all the letters between the two flanking letters.

For example:

- "A-Z" for all uppercase letters,
- "a-z" for all lowercase letters,
- "z-a" for all lowercase letters in reverse order and
- "0-9" for all digits.

The only disadvantage is that the minus sign itself has to be written as "---", so as not to be confused as a range operator.

In the detailed representation of the alphabets (click on the "..." -button), the alphabets can be edited in the short-write mode.

Multiple alphabets

s possible to distinguish between 2 types of actions in the plain text: uppercase letters [A-Z] and gits [0-9]

1. Complete alphabet:
   The two partial alphabets [A-Z] and [0-9] are combined in a certain order to form a new alphabet [A-Z0-9]. If you 'shold' here 3, you get to the '2'.
2. Separated partial alphabets:
   However, all transformations within the respective sub-alphabet are also carried out. This means that a capital letter in the plain text is also mapped to a capital letter in the encrypted text. If you 'shuffle' here 3, you get to the 'C'.

Separated partial Alphabets

The use of several alphabets does not require the algorithms to distinguish between upper and lower case letters.

The algorithm memorizes the alphabet with which it has determined the number of the plaintext. The same alphabet is used to generate the encrypted text.

When a letter occurs in several alphabets, the first of these alphabets is used.

Options regulate the case when a letter does not appear in any alphabet: it is not encrypted, but transferred directly to the output. This is also the case when the letter is in the key. Alternatively, the non-alphabet letters in the key and the plain text can also be filtered out to increase the security. This, however, limits readability.

Example for several alphabets: Addition at Vigenère

Here both approaches are treated: for separate partial alphabets and for a memorized alphabet.

As a small example we consider Vigenère with the following two alphabets:

- L_1 = `"0-9A-F"`
- L_2 = `"0-9a-f`.

In both cases, both the plaintext and the key should both consist of the text `"0123456789abcdABCD"`.

For separate partial alphabets the following results:

- The encrypted text is the smallest digit of an addition of plaintext and key when both are hexadecimal digits.
- The encrypted text is: `"02468ACE02468a468A"`.

- Since the first alphabet L_1 has been used for the digits, digits and uppercase letters in the encrypted text are always the numbers in the plain text.
- Note the difference in 'D' and 'd': The index value is the same, but the 'd' is L_2, so the results differ in the encrypted text: 'A' and 'a'.

For a merged alphabets, the encrypted text is `"02468ACEacACEae024"`.

The following table shows the calculation for the case of the separated partial alphabets L_1, L_2 as well as for a merged alphabet L = `"0-9A-Fa-f"`.

		separate alphabets				**merged alphabet**			
C	*key*	*ind(C)*	*ind(key)*	*ind(out)*	*out*	*ind(C)*	*ind(key)*	*ind(out)*	*out*
0	0	0	0	0	0	0	0	0	0
1	1	1	1	2	2	1	1	2	2
2	2	2	2	4	4	2	2	4	4
3	3	3	3	6	6	3	3	6	6
4	4	4	4	8	8	4	4	8	8
5	5	5	5	10	A	5	5	10	A
6	6	6	6	12	C	6	6	12	C
7	7	7	7	14	E	7	7	14	E
8	8	8	8	16 = 0	0	8	8	16	a
9	9	9	9	18 = 2	2	9	9	18	c
a	a	10	10	20 = 4	4	16	16	32 = 10	A
b	b	11	11	22 = 6	6	17	17	34 = 12	C
c	c	12	12	24 = 8	8	18	18	36 = 14	E
d	d	13	13	26 = 10	a	19	19	38 = 16	a
A	A	10	10	20 = 4	4	10	10	20	e
B	B	11	11	22 = 6	6	11	11	22 = 0	0
C	C	12	12	24 = 8	8	12	12	24 = 2	2
D	D	13	13	26 = 10	A	13	13	26 = 4	4

Partial alphabets: `"0-9A-F"`, `"0-9a-f"`;

Merged alphabet: `"0-9A-Fa-f"`

cleartext = key = "0123456789abcdABCD"

Method 1: Separated: In each sub-alphabet, mod 16 is calculated (hex addition), since each sub-alphabet contains 16 elements, and it remains in the same partial alphabet from which the plaintext letter originates.

Method 2: Merged: In the alphabet, mod 22 is calculated because the alphabet contains 22 elements.

Referenzen

[1] Alphabet (German): https://de.wikipedia.org/wiki/Alphabet_(Kryptologie)

Hill

- Cipher
- Description
- Background
- Security

Plaintext:
```
using hilll cipher
this implenetation
```

Encrypted text:
```
umwut  knzn t lgj
mpm k lsa g nbpel
```

Key matrix:
- ○ 2x2
- ◉ 3x3

```
11 2 3 1 19 18 10 23 4
```

Generate new random key

Options:
- ☐ filter whitespace characters
- ☑ group 5 characters
- ☐ filter non-alphabet characters
- ☐ convert to first alphabet

Details for the Message Alphabets

(chars)
×

Compressed alphabet notation:

Offset:

Keyword for alphabet:

Reverse | Permutate | Rotate left | Rotate right | Transform to other case | Add A–Z | Add a–z | Add 0–9 | Delete alphabet

The Hill cipher was the first cipher purely based on mathematics (linear algebra).
To encipher a message, first the plaintext is broken into blocks of n letters which are converted to numbers, where A=0, B=1, C=2 ... Y=24, Z=25 (so each character is assigned to a number which is usually from the range of 00-25 for the characters A-Z. Upper case and lower case characters are treated equally).
Then the encryption is done by multiplying the numbers with an n x n key matrix modulo 26 (if we have A-Z as our alphabet). The result is converted back to text producing the ciphertext.

Let's assume that we want to encode the message "ACT" with the key "GYBNQKURP".[1]
Since G=6, Y= 24, B=1 etc. we get the following matrix for the chosen key:

$$\begin{pmatrix} 6 & 24 & 1 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{pmatrix}$$

The message is thus encoded by this vector:

$$\begin{pmatrix} 0 \\ 2 \\ 19 \end{pmatrix}$$

Key and message are multiplied with each other and apply modulo 26 to the result:

$$\begin{pmatrix} 6 & 24 & 1 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{pmatrix} \begin{pmatrix} 0 \\ 2 \\ 19 \end{pmatrix} = \begin{pmatrix} 67 \\ 222 \\ 319 \end{pmatrix} \equiv \begin{pmatrix} 15 \\ 14 \\ 7 \end{pmatrix} \pmod{26}$$

This result (15, 14, 7) can be decoded by "POH" which would be the output of the Hill cipher for the chosen message and the used key. To decode the message, one would have to multiply the ciphertext with the inverse matrix of the key and apply modulo 26 to the result.

Details:

The key has to be chosen in such a way that there exists an inverse matrix for the key matrix because it would be impossible to decode the message otherwise. Therefore the determinant of the key matrix modulo 26 has to be co-prime to 26. Numbers co-prime to 26 are: 1,3,5,7,9,11,15,17,19,21,23,25. The determinant of the key matrix shown above is therefore calculated as such:

$$\begin{vmatrix} 6 & 24 & 1 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{vmatrix} \equiv 6(16 \cdot 15 - 10 \cdot 17) - 24(13 \cdot 15 - 10 \cdot 20) +$$

$$1(13 \cdot 17 - 16 \cdot 20) \equiv 441 \equiv 25 \pmod{26}$$

Some implementations like http://massey.limfinity.com/207/hillcipher.php only allow modulo values which are primes. This is better for security but no requirement of the original method. If a length like 26 is used, then this website complains e.g. "Hill cipher won't work unless the alphabet length is prime." This extra requirement can be achieved by adding e.g. an underscore as the first letter.
Implementations without this additional restriction and with the possibility to choose matrix dimensions n other than 2 or 3 are: CrypTool 1, CrypTool 2, and SageMath.

(1) This sample is taken from en.wikipedia.org/wiki/Hill_cipher, 2017-06-05

The Hill cipher was invented in 1929 by Lester S. Hill (*1891; † 1961)who described its method in the journal American Mathematical Monthly (Issue 36).[1]

en.wikipedia.org/wiki/Hill_cipher

The cipher is based on linear algebra only. When parts of the plaintext are known, an attacker could try to find out the key by using a system of linear equations.
So unfortunately, the basic Hill cipher is vulnerable to known-plaintext attacks. An opponent who intercepts n&sup2, plaintext/ciphertext character pairs can set up a linear system which can (usually) be easily solved.