Hexafid Field Cipher v0.7.5 - Cipher Block Chaining (CBC) Mode

A post-apocalyptic field cipher combining classical and digital constructs

Objective: Design a cipher that works easily with pen and paper, secures confidentiality of information, offers plausible deniability if discovered, and exhibits greater strength implemented in software

Note: CBC Mode can only be semantically secure beyond one block with a random Initialization Vector (IV); Recommended use is only for short messages with short lifespans and rotated unique keys

Key Square (K)

M	y	P	a	s	w	o	r
d	1	2	3	A	B	C	D
E	F	G	H	I	J	K	L
N	O	Q	R	S	T	U	V
W	X	Y	Z	b	c	e	f
g	h	i	j	k	l	m	n
p	q	t	u	v	x	z	0
4	5	6	7	8	9	+	/

e.g. K[i] = 'S'

Sequence Addresses (Ks)

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

e.g. Ks[29] = 'S'

Quadtree Addresses (aka Quadresses; Kq)

000000	000001	000100	000101	010000	010001	010100	010101
000011	000010	000111	000110	010011	010010	010111	010110
001100	001101	001000	001001	011100	011101	011000	011001
001111	001110	001011	001010	011111	011110	011011	011010
110000	110001	110100	110101	100000	100001	100100	100101
110011	110010	110111	110110	100011	100010	100111	100110
111100	111101	111000	111001	101100	101101	101000	101001
111111	111110	111011	111010	101111	101110	101011	101010

e.g. Kq[011111] = 'S'

Ks

1	..
..	64

<-- pattern for sequence addresses

pattern for quadtree addresses -->

Kq

00	01
11	10

Original: Keep it secret! Keep it safe!

Plaintext: Keep/it/secret/Keep/it/safe

IV: w3Der4LeOd <-- a set of random characters equal in length to block size (10)

Block 1											Block 2										Block n										
K	**e**	**e**	**p**	**/**	**i**	**t**	**/**	**s**	**e**		**c**	**r**	**e**	**t**	**/**	**K**	**e**	**e**	**p**	**/**		**i**	**t**	**/**	**s**	**a**	**f**	**e**	**P**	**P**	**P**	<-- Plaintext
23	39	39	49	64	43	51	64	5	39		38	8	39	51	64	23	39	39	49	64		43	51	64	5	4	40	39	3	3	3	
w	**3**	**D**	**e**	**r**	**4**	**L**	**e**	**O**	**d**	-->	**L**	**C**	**I**	**8**	**1**	**k**	**j**	**1**	**u**	**7**	-->	**v**	**J**	**c**	**T**	**F**	**f**	**B**	**t**	**s**	**s**	<-- Previous
6	12	16	39	8	57	24	39	26	9		24	15	21	61	10	45	44	10	52	60		53	22	38	30	18	40	14	51	5	5	Ciphertext
29	51	55	24	8	36	11	39	31	48		62	23	60	48	10	4	19	49	37	60		32	9	38	35	22	16	53	54	8	8	
S	**t**	**z**	**L**	**r**	**Z**	**2**	**e**	**U**	**n**		**9**	**K**	**7**	**n**	**1**	**a**	**G**	**p**	**b**	**7**		**V**	**d**	**c**	**Y**	**J**	**D**	**v**	**x**	**r**	**r**	<-- Roundtext
0	1	1	0	0	1	0	1	0	1		1	0	1	1	0	0	0	1	1	1		0	0	1	1	0	0	1	1	0	0	
1	1	0	1	1	1	0	0	1	0		0	1	1	0	0	0	0	1	0	1		1	0	0	1	1	1	0	0	1	1	
1	1	1	1	0	0	0	0	1	0		1	1	1	0	0	0	1	1	0	1		1	0	1	0	1	0	1	1	0	0	
1	0	0	0	1	1	1	1	0	1		1	0	0	1	0	1	0	1	0	0		0	0	0	1	1	1	1	1	1	1	
1	0	0	0	0	0	1	0	1	1		1	0	1	1	1	0	0	0	0	1		1	1	0	0	0	1	0	0	0	0	
1	0	0	1	1	1	1	0	1	0		0	0	0	0	0	1	0	0	0	0		0	1	1	0	1	0	0	1	1	1	
L	**C**	**I**	**8**	**1**	**k**	**j**	**1**	**u**	**7**	-->	**v**	**J**	**c**	**T**	**F**	**f**	**B**	**t**	**s**	**s**	-->	**E**	**h**	**I**	**t**	**v**	**2**	**4**	**P**	**3**	**m**	<-- Ciphertext
0	1	1	0	0	1	0	1	0	1		1	0	1	1	0	0	0	1	1	1		0	0	1	1	0	0	1	1	0	0	
1	1	0	1	1	1	0	0	1	0		0	1	1	0	0	0	0	1	0	1		1	0	0	1	1	1	0	0	1	1	
1	1	1	1	0	0	0	0	1	0		1	1	1	0	0	0	1	1	0	1		1	0	1	0	1	0	1	1	0	0	
1	0	0	0	1	1	1	1	0	1		1	0	0	1	0	1	0	1	0	0		0	0	0	1	1	1	1	1	1	1	
1	0	0	0	0	0	1	0	1	1		1	0	1	1	1	0	0	0	0	1		1	1	0	0	0	1	0	0	0	0	
1	0	0	1	1	1	1	0	1	0		0	0	0	0	0	1	0	0	0	0		0	1	1	0	1	0	0	1	1	1	
S	**t**	**z**	**L**	**r**	**Z**	**2**	**e**	**U**	**n**		**9**	**K**	**7**	**n**	**1**	**a**	**G**	**p**	**b**	**7**		**V**	**d**	**c**	**Y**	**J**	**D**	**v**	**x**	**r**	**r**	<-- Roundtext
29	51	55	24	8	36	11	39	31	48		62	23	60	48	10	4	19	49	37	60		32	9	38	35	22	16	53	54	8	8	
w	**3**	**D**	**e**	**r**	**4**	**L**	**e**	**O**	**d**	-->	**L**	**C**	**I**	**8**	**1**	**k**	**j**	**1**	**u**	**7**	-->	**v**	**J**	**c**	**T**	**F**	**f**	**B**	**t**	**s**	**s**	<-- Previous
6	12	16	39	8	57	24	39	26	9		24	15	21	61	10	45	44	10	52	60		53	22	38	30	18	40	14	51	5	5	Ciphertext
23	39	39	49	0	43	51	0	5	39		38	8	39	51	0	23	39	39	49	0		43	51	0	5	4	40	39	3	3	3	
K	**e**	**e**	**p**	**/**	**i**	**t**	**/**	**s**	**e**		**c**	**r**	**e**	**t**	**/**	**K**	**e**	**e**	**p**	**/**		**i**	**t**	**/**	**s**	**a**	**f**	**e**	**P**	**P**	**P**	<-- Plaintext

Block Padded

Ciphertext: LCI81kj1u7vJcTFfBtssEhItv24P3m <-- with encrypted Block Padding (point d.)

Embedded IV

Ciphertext: w3Der4LeOdLCI81kj1u7vJcTFfBtssEhItv24P3m <-- IV is prepended as first block

Final B64 Padded

Ciphertext: w3Der4LeOdLCI81kj1u7vJcTFfBtssEhItv24P3m <-- no Base64 Padding in this example (point a.)

Influences and References

Classical ciphers - Bifid/Trifid - Delastelle 1902 (for fractionated substitution)

Information Theory - A Mathematical Theory of Communication - Shannon 1948 (for bits and entropy)

Quadtree Data Structures - Finkel and Bentley 1974 (for Key Square addressing)

Six Bit Character Encoding - PEM Standard 1987 (for the origins of Base64 character set) and RFC 4648

Cipher Block Chaining - Message verification and transmission error detection by block chaining - Ersham et al. 1976

 # Open Source: MIT License - https://gitlab.com/hexafid/hexafid

 # Copyright (c) 2020 - 2022 h3ky1

Initial Concept: 2020-02-25

Latest Update: 2022-06-18

Key Setup

Step 1. Draw an 8x8 Key Square with 64 cells

Step 2. Populate the Key Square randomly and uniquely with the characters A-Z a-z 0-9 +/

Key Square Lookup

Step 3. Reference a Sequence Address (e.g. 1..64) for each character in the Key Square

- a. Each character in the Key Square is given a consecutive sequence number, cell by cell then row by row

Step 4. Reference a Quadtree Address (i.e. quadress) for each character in the Key Square (e.g. 010101)

- a. The first 2 bits represent one of the top Key Square quadrants addressed clockwise as 00,01,10,11 then
- b. The second 2 bits represent the middle subdivision of that quadrant into fours, addressed the same way; and
- c. The third 2 bits represent a final bottom subdivision, in the same way; arriving at a 6 bit binary number

Plaintext Encryption

Step 5. Remove non-key characters from the message

- a. Note: With no space in the Base64 character set, plaintext words can be joined together or separated by /

Step 6. Choose a random set of characters, as the IV, equal in length to the block size (10)

Step 7. Separate plaintext into blocks of period size (i.e. 10 characters)

Step 8. Append Block Padding at the end of the block set (see Padding Functions below)

- a. Note: the plaintext, inclusive of the IV, must be padded to both mod 10 = 0 AND mod 4 != 1

Step 9. Add the current plaintext block to the previous ciphertext block (or IV for the first block)

- a. Note: using each character's Sequence Address and modular addition (64)

Step 10. Convert the resulting integers into roundtext characters using the Key Square's sequence addresses

R Step 11. Substitute resulting roundtext characters into matching quadresses using the Key Square's quadtree addresses

O - a. Note: written vertically in columns underneath each character

U Step 12. Transpose substituted quadress within each block

N - a. Note: read horizontally in rows, 6 bits at a time within one block at a time

D Step 13. Decode transposed quadresses into ciphertext characters using the Key Square

Step 14. Assemble blocks together again as ciphertext

Step 15. Prepend ciphertext assemblage with previously chosen IV

- a. Note: only encryption generates an IV; by prependeding to ciphertext, it becomes available for decryption

Step 16. Append Base64 Padding at the end for full ciphertext (see Padding Functions below)

Ciphertext Decryption

Step 5. Remove any non-key characters from the message

- a. Note: this also removes any Base64 Padding

Step 6. Identify the IV from within the ciphertext (ie it should be prepended as the first block)

Step 7. Separate ciphertext into blocks of period size (i.e. 10 characters)

R Step 8. Substitute ciphertext characters into matching quadresses using the Key Square

O - a. Note: written horizontally in rows, 6 bits at a time within one block at a time

U Step 9. Transpose substituted quadresses within each block

N - a. Note: read vertically in columns underneath each character

D Step 10. Decode transposed quadresses into roundtext characters using the Key Square's quadtree addresses

Step 11. Convert roundtext characters into integers using the Key Square's sequence addresses

Step 12. Subtract the previous ciphertext block (or IV for the first block) from current roundtext block

- a. Note: using each character's sequence address and modular subtraction (64)

Step 13. Remove padding equal to the sequence number of last character

Step 14. Throw out the included IV as it is no longer necessary and should never be reused

Step 15. Assemble the blocks together again as plaintext message

Step 16. Optionally restore any non-key characters into message for readability (e.g. spaces)

Block Padding - The plaintext should be a multiple of the block size

Pre-Encryption. Add padding to end of plaintext before encryption

- a. Determine pad length by subtracting length of last block from block size (e.g. 10 - 7 = 3)
- b. If pad length is zero, add a full block of padding
- c. Choose pad character from the Key Square at sequential position equal to pad length (e.g. Ks[3] = 'P')
- d. Add pad length of pad characters to end of plaintext message

Base64 Padding - The ciphertext should appear encoded as Base64

Post-Encryption. Add padding to full ciphertext after encryption

- a. If len(ciphertext) mod 4 = 0 then pad nothing
- b. If len(ciphertext) mod 4 = 1 then review Step 8a. (i.e. should not occur)
- c. If len(ciphertext) mod 4 = 2 then pad ==
- d. If len(ciphertext) mod 4 = 3 then pad =