SNIPH

Snerx's N-Dimensional Cipher

Sniph is meant to be a dynamically shifting multi-dimensional cipher. The impulse for creating this cipher was twofold: to create the first dynamically scalable N-dimensional cipher and to create the first cipher that is *in principle* impossible to mathematically brute force. Further, I wanted to be able to use a homebrewed cipher without higher-maths so the entire process could be easily done by hand. This is a work in progress but all these goals are now provably met, and so if any changes occur from now on, they will be minor changes.

The way it works is by generating tables of variable sizes that are then seated inside of each other to the N*th* dimension specified by the user in the cli. The breadcrumb or 'pathway' through each dimension/table-set is chosen based on a passphrase. For example, if your passphrase was 'henlo', the cipher would look at 'h' as being the 7*th* letter of the alphabet and then pick the 7*th* position in the table to open. The cipher would then look at 'e' as being the 5*th* letter of the alphabet and pick the 5*th* position in the next table to open. This continues until every letter is used, and then the process repeats starting with the first letter again until *n*-2 tables in the path are opened. The reason for it being *n*-2 is explained later.

In the first set of tables to the right, the tables generated are a 3x3 size, and go 3 dimensions deep. Through the highlighted paths, the letter 'H' can be represented by the path of 1-2, then 1-1, and then 3-2. The letter H is then translated into its path: '121132'. Using the right, 'HELLO' looks like '121132121122121213121213121223'. Similarly, in a 5x5 table that only goes 2 dimensions deep, an 'H' could like like '2423' **or** '5312'. Having many outputs code for the same input adds significant obfuscation to the cipher. Note that the last two dimensions (seated table sets) are what determines the character being chosen, so the passphrase used to determine the path up until that point does not determine the table positions of the last two tables. This is to create a separation between the OTP keys and the letter to be deciphered. In these last two tables the letter being chosen will appear multiple times and the choice for which position to pick is determined randomly in the code. This makes it possible to have both true randomness and predetermined OTP keys in the same cipher. To clarify, this allows you to use OTP keys without having to transmit the OTP keys each time you generate ciphertext **and** allows you to reuse the same keys and not generate similar-looking output each time. The reason why the pathways matter so much and why they are being treated as OTP keys is because the most significant feature of this cipher is the dynamically generated 'shift' or character translation determined by the path to each character. Every two numbers in a ciphered character's path, with the exception of the last four (since those are used to determine specific table spots), is taken as an x-y equation, and these are added up to determine the total number of characters to shift (randomly gen'd Caesar cipher). For example, the charset from those first 3x3 tables, after choosing 1-2 (which results in -1), would be shifted back one character. This means that the pathway for each character makes each character carry its own one-time-pad, arbitrarily generated by a passphrase.

Additionally, the numbers chosen for paths are not limited to the dimensions of the tables they are generated from. To stop text-analysis from revealing the table size (which would be obvious if the only numbers used were 1, 2, and 3, like in the first table set on the right), all numbers from 0 to 9 are used. The leftover numbers wrap and are re-used in sequence as duplicates of the initial numbers. For example, after 3, 4 could stand in for 1, 5 for 2, 6 for 3, and 7 for 1 again, etcetera. Given this, the last 'H' shown on the right, stated as '5312', could also appear as '0817' or '5862'. Selecting which numbers to use, the default or a leftover, is randomnly decided by the code at no extra processing cost since this is arbitrary for ciphering the text. This step makes discovering the size of the table used impossible by simply looking at the ciphered text. In a similar vein, the dimensionality of the ciphered text is made impossible to discover- the output cuts out the initial pathway numbers since those are determined by the passphrase. Each input character then has four output characters, specifically the four that determine the positions of the character in the last two tables. Further, knowing two of the three input variables still does not let you break the cipher since without the third you will get nothing but junk text as output every time. As an extra layer of security, text at the end can be shifted over and wrapped around into the front of the message, or vice versa. In the cli this would be the 'Offset Variable'.

Ciphering text with Sniph comes with very little processing cost since no higher maths are used, pathways to characters are arbitrary, and the disproportionate majority of tables do not need to be generated. The processing cost to decipher however, if you don't know the table size or dimensions or passphrase/OTP keys, is far more than just an exponential jump. Every possible table size and dimension combination has to be generated and tried in order to attempt a brute-force of ciphered text under Sniph. Becuase N-dimensions means *N-dimensions*, you can set it to 1,000-D with 10x10 tables, and the random-looking outputs it would generate due to each character carrying its own OTP key makes brute-forceing this create a hermeneutical lacuna.

Sniph has a minor limitation- the smallest table size and dimensionality is 3x3 and 4, respectively. This is due to the size of the charset shown on the right. Since the tables are already technically 2-dimensional, and a minimum of 3 extended dimensions are added on to the base tables in Sniph, this makes Sniph start as a 5D cipher. To my knowledge, the highest dimension cipher in commercial use is in a 5D encryption protocol. Considering that Sniph **starts** at 5D, this should make it stand out.

First Dimension

	1	2	3
1	1-1	1-2	1-3
2	2-1	2-2	2-3
3	3-1	3-2	3-3

Second Dimension

1-1	1	2	3
1	1-1	1-2	1-3
2	2-1	2-2	2-3
3	3-1	3-2	3-3

1-2	1	2	3
1	1-1	1-2	1-3
2	2-1	2-2	2-3
3	3-1	3-2	3-3

Third Dimension

1-1	1	2	3
1	A	B	C
2	D	E	F
3	G	H	I

1-2	1	2	3
1	J	K	L
2	M	N	O
3	P	Q	R

First Dimension

	1	2	3	4	5
1	1-1	1-2	1-3	1-4	1-5
2	2-1	2-2	2-3	2-4	2-5
3	3-1	3-2	3-3	3-4	3-5
4	4-1	4-2	4-3	4-4	4-5
5	5-1	5-2	5-3	5-4	5-5

Second Dimension

1-1	1	2	3	4	5
1	A	B	C	D	E
2	F	G	H	I	J
3	K	L	M	N	O
4	P	Q	R	S	T
5	U	V	W	X	Y

5-3	1	2	3	4	5
1	G	H	I	J	K
2	L	M	N	O	P
3	Q	R	S	T	U
4	V	W	X	Y	Z
5	.	,	?	!	:

Note On The Above Tables

These were not generated with the cipher itself, they are merely for representational purposes. The actual cipher needs to be at least four seated dimensions deep, so the text in the above tables is arbitrarily thrown in for example purposes only. Also, the table sides do not need to be the same length, they can be 3x7, or 8x2, or 10x9, etcetera.

Minimal Character Set

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	2	3	4	5	6	7
8	9	0	space	newline	case-shift	70 total	

https://github.com/VivaCaligula/SNYPH

Sniph is based on an older project of ours under a very similar name, the source code of which is hosted in the above link.

2018/2/7