**G2GA1 Cipher**

The G2GA1 Cipher is a polygraphic substitution cipher that uses three rounds, three keys, a key derivation function, and a stochastic algorithm to encrypt/decrypt messages.

Design overview

The G2GA1 Cipher uses three keys to encrypt and decrypt any message written in characters from the monocase English alphabet.

The cryptosystem consists of a key derivation function (KDF,) an encryption function, and a decryption function. The encryption and decryption algorithms both use the same KDF for Round 1.

The KDF generates a matrix from the characters in the first key. The number of columns in the matrix is specified by the second key value. The number of rows will be dependent on the second key and length of the first key.

The encryption algorithm uses three rounds to encrypt the message. The first round, Round 1, uses the matrix generated in the KDF, and finds a random occurrence of each character from the plaintext message in the matrix. A set of ordered pairs are returned from Round 1. Round 2 takes the ordered pairs and transforms them into an encoding using the monocase English alpabet. The final step, Round 3, uses the third key to encrypt the output from Round 2 with the Vigenère Cipher.

The decryption algorithm uses an inverse version of each round from the encryption algorithm in the reverse order. Round 1 uses the third key to decrypt the Vigenère Cipher's ciphertext and produce the encoded string of ordered pairs. This string of ordered pairs is decoded in Round 2. Round 3 reads each ordered pair and performs a coordinate lookup in the matrix, from the KDF, and generates plaintext based on the corresponding entries.

Design goals

1. Use cryptography methods introduced prior to the midterm.
2. Defend against cryptanalysis methods introduced prior to the midterm.
3. Design will be able to encrypt and decrypt a message of no less than 1kB in less than five minutes wall clock time.
4. Design will not include any binary operations.
5. Design will not use any 'known hard problems' from modern cryptography.

Design features

1. A key derivation function is used to defend against partial discovery of keys.
2. Message letter frequency distribution is flattened to prevent any plaintext characterization leaking into the ciphertext.
3. A injective mapping is used between message plaintext and glyphs to force any autocorrelation to zero.
4. A stochastic approach is taken to ensure that brute force attempts fail (the same message/keys will not produce the same ciphertext.)

Suspected weaknesses

//

Key Derivation Function

The KDF generates a matrix using the characters from the first key as entries. The number of columns in the matrix is determined by the second key, and the number of rows will be dependent on both the second key and the length of the message. If the length of the message is not divisible by the second key then the final row will be padded by the required amount of characters form the beginning of the key.

During encryption, the KDF also verifies that first key contains at least one occurrence of each character in the message (the function mapping characters from the message to the first key is surjective.)

Encryption

*G2GA1 Ciphertext = G2GA1-Encrypt(****message****,* ***k1****,* ***k2****,* ***k3****)*

* All string data is processed in monocase English.
* **message**: A string containing the plaintext to encrypt.
* **k1**: A string containing the first key.
* **k2**: An integer, containing the second key with the following constraint:
  + 0 < **k2** =< length (**k1**).
* **k3**: A string containing the passphrase used to encrypt the encoded ordered pair string from Round 3.

Pre-Round 1 “Matrix generation”

The KDF uses **k1** and **k2** to generate it's matrix.

Round 1 “Stochastic plaintext mapping”

Round 1 of the encryption algorithm maps each character from the plaintext message to an entry in the matrix from generated by the KDF. The ordered pair (row, column) from the *n*'th occurrence of the plaintext character in the matrix. The integer value, *n*, is chosen at random and contained by 1 <= *n* =< length(**k1**). The mapper will start over at the top of matrix if the *n* is great than the number of occurrences of any plaintext character. The output of Round 1 is a string containing all the ordered pairs.

An example of Round 1 output: “234,23,0,22,44,34,234,1” which would be four plaintext characters, in the KDF matrix, represented by the following set { (234,23), (0,22), (44,34), (234,1) }.

Round 2 “Coordinate encoding”

The ordered pairs string from Round 1 will be encoded based on the following rules:

‘0’ -> ‘A’ ‘5’ -> ‘G’

‘1’ -> ‘B’ ‘6’ -> ‘H’

‘2’ -> ‘C’ ‘7’ -> ‘I’

‘3’ -> ‘D’ ‘8’ -> ‘J’

‘4’ -> ‘E’ ‘9’ -> ‘K’

Encode each element's delimiter, designated by a ',', with by a randomly chosen character from ’L’ to 'Z'.

Round 2 Example:

Input: “234,23,0,22,44,34,234,1”

Output: “CDELCDUARCCXEEZDEQCDELB”

Round 3 “Vigenère Cipher encryption”

Round 3 encrypts the string from Round 2 with the third key, **k3**, to produce the output for the encryption algorithm.

Decryption

*Plaintext = G2GA1-Decrypt(****ciphertext****,* ***k1****,* ***k2****,* ***k3****)*

* All string data is processed in monocase English.
* ***ciphertext***: A string containing the ciphertext to decrypt.
* **k1**: A string containing the first key.
* **k2**: An integer, containing the second key with the following constraint:
  + 0 < **k2** =< length (**k1**).
* **k3**: A string containing the passphrase used to decrypt the encoded ordered pair string from Round 1.

Pre-Round 1 “Matrix generation”

The KDF uses **k1** and **k2** to generate it's matrix.

Round 1 “Vigenère Cipher decryption”

Round 1 decrypts the **ciphertext** with the third key, **k3**, to produce the ordered pairs string used in Round 2.

Round 2 “Coordinate decoding”

The ordered pairs string from Round 1 will be decoded based on the following rules:

‘A’ -> ‘0’ ‘F’ -> ‘5’

‘B’ -> ‘1’ ‘G’ -> ‘6’

‘C’ -> ‘2’ ‘H’ -> ‘7’

‘D’ -> ‘3’ ‘I’ -> ‘8'

‘E’ -> ‘4’ ‘J’ -> ‘9’

Each element is delimited by a character from ’L’ to 'Z' and should be decoded as a ','.

Round 2 Example:

Input: “CDELCDUARCCXEEZDEQCDELB”

Output: “234,23,0,22,44,34,234,1”

Round 3 “Plaintext lookup”

Round 3 processes the string of ordered pairs produced by Round 2. Each ordered pair is pulled from the string and the corresponding plaintext character is found at the coordinate in the matrix produced by the KDF. The plaintext characters are stored in aggregate and the final plaintext string is the output of the decryption algorithm.

Full encryption/decryption example

//