Vigenere Cipher in Lisp

gene m. stover

created Tuesday, 2004 August 3
updated Monday, 2006 April 17

1 What is the Vigenere Cipher?

Vigenere is a symmetric, polyalphabetic substituion cipher. The "substitution" part of that label means that it encrypts by replacing characters. The "polyalphabetic" part of that label means that it uses a different ciphertext alphabet for each character it encrypts – until it runs out of alphabets & re-uses the first one again. The "symmetric" part means that it uses the same key for encryption & decryption.

Here's an example:

1. Let the plaintext be "hello world". (The plaintext alphabet is all twenty-six case-insensitive letters written as lower case. The ciphertext alphabet is all twenty-six case-insensitive letters written as upper case.)

2. Let the encryption key be "vig" (as in *Vig*enere).

3. The first character from the plaintext, "h", is encrypted with the first character from the key, "v". *H* is letter number seven from the alphabet, *V* is the 22nd letter in the alphabet, so we add the 21 from *V* to the seven from *H*, giving us 28. We map 28 back into the alphabet with a *mod 26*, which is two. Letter number two of the alphabet is *C*. So the first letter of the ciphertext is *C*.

4. Encrypt the second letter of the plaintext, "e", with the second letter of the key, "i", in the same way. *E* is the fifth letter of the alphabet, *I* is the ninth, so the next ciphertext letter is $5 + 9(mod 26) \rightarrow 14 \rightarrow N$.

5. Encrypt the third plaintext letter, "l" with the third key letter, "g", giving *S*.

6. Four the fourt plaintext letter, "l", we're out of key characters, so we cycle back to the beginning of the key. So we encrypt the fourt plaintext letter with the first key letter. *V* encrypts *L* to *F*.

Notice that we've encrypted L twice, but the first L encrypted to S, while the second L encrypted to F.

7. Continue encrypting the rest of the letters of the plaintext in the same way.
8. The final ciphertext is "CMRGW RWXGL".

Decrypt Vigenere ciphertext by using the key to tell how to shift ciphertext letters backwards in the alphabet, giving the plaintext letters.

Vigenere is described with more skill than I have in [?], [?], & [?].

2 Security of Vigenere

The Vigenere cipher is an old cryptosystem from the days of pen & paper. Because it was designed to be secure against pen & paper, it wouldn't hold up for a microsecond against modern professional cryptanalysis software on fast computers, & it might not hold up that long against cryptanalysis on a home computer. Vigenere is not sufficient for security in a modern, computerized environment.[1]

3 What's Interesting about My Implementation

The Vigenere cipher itself is probably of interest & use only to people who care about the history of cryptology. So why should you be interested in this implementation of the Vigenere cipher?

I can't honestly say that you *should* be interested, but I think I did a few things that are worth noticing.

3.1 Parameterized Alphabet

The main cool thing is the alphabet parameter to the encryption & decryption functions. Most encryption & decryption functions assume that you are encrypting octets. Maybe that's fine most of the time, but it would be good to choose the alphabet sometimes. I think all encryption & decryption software should allow you to specify the alphabet, if appropriate.

3.2 Circular Lists

Encryption & decryption loop over the key. When the loop runs out of letters in the key, it returns to the first letter in the key.

The most common way to do this is probably to store the key in a vector & to put a range check around the index after it is incremented.

[1] And the fact that I'm telling this means that I'm aware of it, so I don't need people to e-mail me to let me know that Vigenere is old & that there are more secure cryptosystems to be had now. Duh! Everyone wants to demonstrate that he's cryptosavvy.

I chose to store the key as a circular list so the CDR of the last letter in the key is the first letter of the key. In other words, CDR always returns the correct value, with no need to do arithmetic on a vector's index.

This trick helped shorten the loop, but the initialization code (in function MAKE-ALPHABETS) is more complex than the initialization code for an encryption function that assumes the characters are octets. In fact, I dislike the initialization code. Neither do I like how I implemented decryption; I think there could be a cleaner way.

4 The Complete Source Code

The source code is at ./vigenere.lisp[2] . I release it under the terms of *The Gnu Lesser General Public License* ([**?**]).

5 Usage

My implementation of Vigenere is in just one source file, `vigenere.lisp`.

Load it into your Lisp with an expression like this:

```
> (load "vigenere.lisp")
T
```

The single function for encryption is VIGENERE-ENCRYPT. It's two required arguments are the plaintext & the key. It has a third argument, the alphabet, but we'll discuss that in Section 5.1.

The plaintext & the key must be sequences, but they may be any kind of sequence, namely lists or vectors. There are no restrictions on the elements within them as long as they are in the alphabet. The alphabet defaults to alphanumeric characters, so we'll use that for now.

Here's an example of encryption & decryption:

```
> (defvar *plaintext* "hello world")
*PLAINTEXT*
> (defvar *key* "vig")
*KEY*
> (defvar *ciphertext* (vigenere-encrypt *plaintext* *key*))
*CIPHERTEXT*
> *ciphertext*
"MwBQG 1GHQv"
> (vigenere-decrypt *ciphertext* *key*)
"hello world"
> (equal * *plaintext*)
T
```

[2]`http://cybertiggyr.com/gene/vig/vigenere.lisp`

Or, to be really really terse about what's going on:

```
> (equal (vigenere-decrypt
           (vigenere-encrypt
             *plaintext* *key*)
           *key*)
         *plaintext*)
T
```

5.1 Alphabets

Compare the ciphertext from this recent example, "MwBQG 1GHQv", to the one we got in the first example, "CMRGW RWXGL". They're different? What gives?

What gives is that we used different alphabets in the first example encryption & this more recent one. The alphabet from the earlier example consisted of the case-insensitive letters, but the more recent example used default alphabet for VIGENERE-ENCRYPT, which is the case-sensitive letters & the numbers.

I can use my VIGENERE-ENCRYPT & VIGENERE-DECRYPT functions to reproduce the first example if I also specify an appropriate alphabet. Here's how:

To specify a case-insensitive alphabet, I'll create an alphabet of all the letters in one case. I choose upper case because it reminds me of what people thought of computers back before 1980. For brevity, I'll make the non-portable, though probably correct, assumption that my computer uses the ASCII character set. I can collect the upper case letters into a list like this:

```
> (defvar *myalphabet*
          (loop for i from (char-code #\A)
                      to (char-code #\Z)
                collect (code-char i)))
*MYALPHABET*
```

The plaintext & key are in lower case, so I must remember to map them to my alphabet case before encrypting it. That's easy enough with STRING-UPCASE. I'll use the *PLAINTEXT* & *KEY* global variables I defined in the previous example.

Here's my VIGENERE-ENCRYPT function used to reproduce the first example:

```
> *plaintext*
"hello world"  ; same as it ever was
> *key*
"vig"          ; same as it ever was
> (setq *ciphertext*
        (vigenere-encrypt
          (string-upcase *plaintext*)
          (string-upcase *key*)
          *myalphabet*))
"CMRGW RWXGL"
```

For convenience, this alphabet of upper case letters is defined in `vigenere.lisp` as *VIGENERE-SIMPLE-ALPHABET*.

If I hadn't mapped the plaintext to upper case, VIGENERE-ENCRYPT wouldn't have encrypted it because I wrote VIGENERE-ENCRYPT to pass characters that aren't in the alphabet without encrypting them. Here's what would have happened it I hadn't converted the plaintext to upper case:

```
> (vigenere-encrypt
          *plaintext*
          (string-upcase *key*)
          *myalphabet*)
"hello world"
```

The key must be in the alphabet, or we'll get an error. Here's an example:

```
> (vigenere-encrypt
          (string-upcase *plaintext*)
          *key*
          *myalphabet*)
crisp runtime: TYPE-ERROR
$
```

Switching between subtle variations of text alphabets only scratches the surface of what can be accomplished with the alphabet parameter.

With the alphabet parameter, VIGENERE-ENCRYPT can operate on octets. Here's an example:

The alphabet is all the positive integral values less than 256:

```
> (defvar *octets*
          (loop for i from 0 to 255 collect i))
*OCTETS*
```

Plaintext is the binary data, maybe from some binary file:

```
> (setq *plaintext* (list 0 1 2 253 254 255))
(0 1 2 253 254 255)
```

The key must be binary, but instead of making our correspondent remember a list of arbitrary byte values, we'll let her remember a keyword, which we'll convert to binary values. (We're assuming that my computer & our correspondent's computer use the same character set.) We could create they key of octet values from a keyword like this:

```
> (setq *key*
        (loop for i across "vig" collect (char-code i)))
(118 105 103)
```

Now let's encrypt it:

```
> (setq *ciphertext*
        (vigenere-encrypt *plaintext* *key* *octets*))
(118 106 105 115 103 102)
```

After my correspondent had set her *OCTETS*, *KEY* & *CIPHERTEXT* global variables, she could decrypt it like this:

```
> (vigenere-decrypt *ciphertext* *key* *octets*)
(0 1 2 253 254 255)
```

. . . which is the same as the original plaintext:

```
> (equal * *plaintext*)
T
```

The alphabet parameter can even be used to encrypt things other than characters & octets. Let's encrypt a list of symbols.

First, let's make an alphabet from a bunch of the usual Lisp symbols. Let's also make a key using those symbols:

```
> (setq *myalphabet*
        (list 'defun '+ '- 'let 'car 'cdr
              nil t 'equal 'foo 'x 'i 'loop
              0 1 2 3 'test 'my 'a 'list))
(DEFUN + - LET CAR CDR NIL T EQUAL FOO X I
 LOOP 0 1 2 3 TEST MY A LIST)
> (setq *key* '(defun foo i let x))
(DEFUN FOO I LET X)
```

Now let's set a plaintext, encrypt it, & test the decryption:

```
> (setq *plaintext* '(let my defun test a list of car))
(LET MY DEFUN TEST A LIST OF CAR)
> (setq *ciphertext*
        (vigenere-encrypt *plaintext* *key*
                          *myalphabet*))
(LET () I LIST EQUAL LIST OF 2)
> (equal *plaintext*
         (vigenere-decrypt
           *ciphertext* *key* *myalphabet*))
T
```

Why would you want to encrypt a sequence of symbols? Heck, I don't know, but you can. Maybe there is some cool trick that could be done with encrypted Lisp source code.

A Vigenere in Pascal

Here is source code for a Vigenere cypher in Delphi Pascal. It's pretty plain; it's not a Delphi component.

I'm not an experienced Pascal programmer, so I apologize for making naïve Pascal mistakes. I suspect I made such a mistake when I used `ArrayOfOctets`.

By the way, you'll need to define your own `ArrayOfOctets` type. Something like "`type ArrayOfOctets = Array of Byte`".

I have actually used this code on a project. I wrote it from scratch, for myself, & it's licensed the same as the Lisp code in this essay.

```
unit GeneraVigenere;
(*
 * Vigenere encryption & decryption on arrays of octets.
 *)

interface

uses GeneraTypes;

(*
 * Decrypt a Vigenere-encrypted binary message. Return the plaintext.
 *)
function Decrypt (ciphertext, key : ArrayOfOctets) : ArrayOfOctets;

(*
 * Encrypt a message via Vigenere & the key. Return the cyphertext.
 *)
function Encrypt (plaintext, key : ArrayOfOctets) : ArrayOfOctets;

implementation

uses SysUtils;

function Decrypt (ciphertext, key : ArrayOfOctets) : ArrayOfOctets;
var
   plaintext : ArrayOfOctets;
   ti : Integer; (* {cipher, plain} Text Index *)
   ki : Integer; (* Key Index *)
begin
   SetLength (plaintext, Length (ciphertext));
   ki := 0;
   for ti := Low (ciphertext) to High (ciphertext) do begin
      plaintext[ti] := (ciphertext[ti] + key[ki]) mod 256;
      ki := ((ki + 1) mod Length (key)) + Low (key)
   end;
   Decrypt := plaintext
end; { Decrypt }

function Encrypt (plaintext, key : ArrayOfOctets) : ArrayOfOctets;
var
   ciphertext : ArrayOfOctets;
   ti : Integer; (* {cipher,plain}Text Index *)
   ki : Integer; (* Key Index *)
begin
   SetLength (ciphertext, Length (plaintext));
   ki := 0;
   for ti := Low (plaintext) to High (plaintext) do begin
      ciphertext[ti] := (plaintext[ti] - key[ki]) mod 256;
      ki := ((ki + 1) mod Length (key)) + Low (key)
   end;
   Encrypt := ciphertext
end; { EncryptVigenere }

end.
```

References