Bad Old Ciphers: Shift Cipher

A shift cipher **encrypts** by shifting each plaintext letter forward by an amount (the **key**) known only to the sender and authorized recipient. Characters shifted off the end are **wrapped around**.

Use the standard 26-letter English alphabet as main example:

```
abcdefghijklmnopqrstuvwxyz
```

For example, shift the **plaintext**

```
my dog has fleas
```

forward by 2 to obtain **ciphertext**

```
OA FQI JCU HNGCU
```

Note that the `y` got pushed off the end and wrapped back to become `a`.

It is common practice to show plaintext in lower-case, ciphertext in upper-case.

An **unauthorized attempt at decryption** is called an **attack**.

That is, an attempt to either **get the key** or **decrypt without the key** is an *attack*.

Leaving punctuation and word-breaks *in* is **bad**, because it makes attacks easier.

To **decrypt** knowing the **key**, simply shift the ciphertext backward by the key amount, wrapping around if letters shift off the *front* of the alphabet.

For example, if the key is 5 and the ciphertext received is

```
MJQQT, BTWQI
```

decrypt by shifting each letter backward by 5 to get

```
hello, world
```

Note that the `B` shifted off the front of the alphabet and wrapped around to become `w`.

Attacking the shift cipher

The shift cipher is **bad** because it has only 25 possible keys, and 25 is a too-small number. In other words, the **keyspace** is too small.

An attacker can easily do the decryption operation with all 25 possible keys, and choose the decryption which makes sense.

This works only if the plaintext is in a natural language, such as English. Unrecognizable language cannot be decrypted this way.

For example, if we intercept ciphertext

```
YUNWCHXOBDWBQRWNJUUMJH
```

we try shifting backward by 1, 2, ...

```
by 1: xtmvbgwnacvapqvmittlig
by 2: wsluafvmzbuzopulhsskhf
by 3: vrktzeulyatynotkgrrjge
by 4: uqjsydtkxzsxmnsjfqqifd
by 5: tpirxcsjwyrwlmriepphec
by 6: sohqwbrivxqvklqhdoogdb
by 7: rngpvaqhuwpujkpgcnnfca
by 8: qmfouzpgtvotijofbmmebz
by 9: plentyofsunshineallday
```

and the last one makes sense in English.

In fact, we did too much work in this attack.

In that example, one could tell that the incorrect decryptions were incorrect by just looking at the first few letters.

It was a waste of effort to decrypt the whole line when the first 6 characters were not likely to be English.

Of course, one might be too conservative in assessing this, and generate **false negatives**, but this is not terrible. You can always go back...

So in trying to decrypt

```
YUNWCHXOBDWBQRWNJUUMJH
```

we should have shifted backward just the first few characters by 1, 2, ...

```
1: xtmvbg
2: wsluaf
3: vrktze
4: uqjsyd
5: tpirxc
6: sohqwb
7: rngpva
8: qmfouz
9: plenty
```

and then finish the last one since it looks promising.

The shift cipher is broken: we can verify that a successful hostile attack takes just slightly more effort than the authorized decryption.

For an N-character plaintext, the authorized decryptor does N shifts to decrypt.

The attacker on the average would do about $13 = 26/2$ shifts of the first 6 characters before finding the key, and then do N shifts. Thus, the attacker does about $N + 78$ shifts.

For N large, this is a tiny difference.

The **workload** *for an attacker must vastly greater than for authorized encryptor and decryptor!*

Letters as Numbers

A fundamental point is that we should translate or encode letters and any other characters as *numbers* to better understand the manipulations involved in encryption and decryption.

For alphabet `abc...yz` translate

$$\begin{gathered} \texttt{a} \to \texttt{0} \\ \texttt{b} \to \texttt{1} \\ \texttt{c} \to \texttt{2} \\ \ldots \\ \texttt{y} \to \texttt{24} \\ \texttt{z} \to \texttt{25} \end{gathered}$$

In principle, it doesn't matter whether we start at 0 or at 1, but we'll start at 0.

Reduction modulo m

The shift-with-wrap-around operation translates nicely into mathematics.

The **reduction modulo** m operation

$$x \,\%\, m$$

is the operation of dividing-with-remainer x by m, throw away the quotient, and keep only the remainder.

For example,

$$\begin{array}{ccccc}
7 & \% & 3 & = & 1 \\
19 & \% & 7 & = & 5 \\
21 & \% & 7 & = & 0 \\
39 & \% & 9 & = & 3 \\
103 & \% & 10 & = & 3 \\
120 & \% & 10 & = & 0
\end{array}$$

Then shifting a character x forward by k translates into

$$x \to (x + k) \,\%\, 26$$

For example, to shift `t` forward by 11, since `t` encodes as 19, translates into

$$\texttt{t} \to 19 \to (19 + 11) \,\%\, 26 = 4 \to \texttt{e}$$

It is a question of **convention** how to divide negative dividends. Our convention is: to divide x by m gives a **remainder** r and **quotient** q with

$$0 \leq r < |m|$$

and

$$x = q \cdot m + r$$

Thus, for **negative** x (m positive)

$$x \,\%\, m = \begin{cases} m - (|x| \,\%\, m) & (|x| \,\%\, m \neq 0) \\ 0 & (|x| \,\%\, m = 0) \end{cases}$$

Example:

$$(-11) \,\%\, 3 = 3 - (11 \,\%\, 3) = 3 - 2 = 1$$

since $11 \,\%\, 3 \neq 0$.

Inverses modulo m

A **(multiplicative) inverse of x modulo m** (both integers) is an *integer* y (if it exists) such that

$$(x \cdot y) \,\%\, m = 1$$

For example, 3 is a multiplicative inverse of 2 modulo 5 because

$$(2 \cdot 3) \,\%\, 5 = 6 \,\%\, 5 = 1$$

For example, 7 is a multiplicative inverse of 3 modulo 10 because

$$(3 \cdot 7) \,\%\, 10 = 21 \,\%\, 10 = 1$$

Also -3 is a multiplicative inverse of 3 modulo 10 because

$$(3 \cdot -3) \,\%\, 10 = -9 \,\%\, 10 = 1$$

Also 17 is a multiplicative inverse of 3 modulo 10 because

$$(3 \cdot 17) \,\%\, 10 = 51 \,\%\, 10 = 1$$

Also 27 is a multiplicative inverse of 3 modulo 10 because

$$(3 \cdot 27) \,\%\, 10 = 81 \,\%\, 10 = 1$$

How do we find these multiplicative inverses?

Eventually we will use the Euclidean algorithm, but for now we do this to illustrate **brute force**: to find the multiplicative inverse of 3 modulo 7:

$$\begin{array}{lcc}
\text{Try 1:} & (3\cdot 1)\,\%\,7 = 3 \neq 1 & \text{no} \\
\text{Try 2:} & (3\cdot 2)\,\%\,7 = 6 \neq 1 & \text{no} \\
\text{Try 3:} & (3\cdot 3)\,\%\,7 = 2 \neq 1 & \text{no} \\
\text{Try 4:} & (3\cdot 4)\,\%\,7 = 5 \neq 1 & \text{no} \\
\text{Try 5:} & (3\cdot 5)\,\%\,7 = 1 & \text{yes}
\end{array}$$

That is, just proceeding systematically but unimaginatively we will inevitably find a multiplicative inverse.

Functions versus formulas

The encoding of letters as numbers is an example of a **function** that is not really given by a **formula**.

Standard notation is that

$$f : A \to B$$

means f is a function from a **set** A to a **set** B, like our encoding

$$f : \{\mathtt{a}, \mathtt{b}, \ldots, \mathtt{z}\} \to \{0, 1, \ldots, 25\}$$

That notation by itself does not tell us how to **compute** f, however. A *computational* description could be by a *formula* or by a **look-up table**.

Recall that a **set** is an *unordered* collection of things.

A set can be described as a comma-separated **list** enclosed by *braces* (though the apparent ordering is *not intrinsic*), like

$$\{1, 2, 3\}$$

Since order does not matter,

$$\{1, 2, 3\} = \{3, 1, 2\} = \{2, 1, 3\} = \text{ etc.}$$

Also, repeating an element does not do anything:

$$\{1, 2, 3\} = \{1, 1, 1, 2, 2, 3, 2\}$$

The things x in a set S are the **elements** of the set. Notation is $x \in S$ or $S \ni x$. Examples:

$$1 \in \{1,2,3\}$$

$$\{1,2,3\} \ni 3$$

$$4 \notin \{1,2,3\}$$

The **union** $A \cup B$ of two sets consists of the elements lying in either set. Example:

$$\{1,2,3\} \cup \{3,4,5\} = \{1,2,3,4,5\}$$

The **intersection** $A \cap B$ of two sets consists of the elements lying in both. Example:

$$\{1,2,3\} \cap \{3,4,5\} = \{3\}$$

Two sets A, B are **disjoint** if they have no elements in common, that is, if their intersection is the empty set

$$\phi = \{\}$$

Sets can have elements which are themselves sets. Example:

$$\{1, 2, 3, \{1, 2\}, \{\{1\}\}\}$$

has elements

$$1, 2, 3, \{1, 2\}, \{\{1\}\}$$

A **look-up table** for a function $f : A \to B$ is a **list** of outputs for all possible legal inputs of f.

(*A function $f : A \to B$* **must***: accept as input every element of the set A, produce the same output for the same input, produce inputs in the set B, and not fail to produce an output.*)

Example: to describe a function

$$f : \{1, 2, 3\} \to \{7, 8\}$$

we must tell exactly 3 things, namely $f(1)$, $f(2)$, and $f(3)$. We do *not* have to give a formula. For example,

$$f(1) = 7 \quad f(2) = 7 \quad f(3) = 8$$

is a legitimate description of *one* particular function f.

We can list *all* functions

$$f : \{1, 2, 3\} \to \{7, 8\}$$

by telling, in each case, the output for every input:

case 1:	$f(1)=7$	$f(2)=7$	$f(3)=7$
case 2:	$f(1)=7$	$f(2)=7$	$f(3)=8$
case 3:	$f(1)=7$	$f(2)=8$	$f(3)=7$
case 4:	$f(1)=7$	$f(2)=8$	$f(3)=8$
case 5:	$f(1)=8$	$f(2)=7$	$f(3)=7$
case 6:	$f(1)=8$	$f(2)=7$	$f(3)=8$
case 7:	$f(1)=8$	$f(2)=8$	$f(3)=7$
case 8:	$f(1)=8$	$f(2)=8$	$f(3)=8$

(The chosen ordering of these 8 functions is *lexicographic* (‘alphabetic’) in terms of the outputs.)

A function $f : A \to B$ is **surjective** (=*onto*) if every element of the *target* set B is *hit* by some element of the *source* set A. That is, for every $b \in B$ there is $a \in A$ such that $f(a) = b$.

Example: the function $f : \{1, 2, 3\} \to \{4, 5\}$ given by

$$f(1) = 4 \quad f(2) = 4 \quad f(3) = 5$$

is surjective because both elements of the target are hit. But

$$f(1) = 4 \quad f(2) = 4 \quad f(3) = 4$$

is *not* surjective because the element 5 in the target is *missed.*

A function $f : A \to B$ is **injective** (=*one-to-one*) if every element of the *target* set B is hit by *at most* one element of the *source* set A. That is, for $a_1, a_2 \in A$ we have $f(a_1) = f(a_2)$ **only** when $a_1 = a_2$.

Example: the function $f : \{1,2\} \to \{4,5,6\}$ given by

$$f(1) = 4 \quad f(2) = 6$$

is injective because no two elements of the source hit the same element of the target. But

$$f(1) = 4 \quad f(2) = 4$$

is *not* injective because the element 4 in the target is *hit twice*.

Counting without listing

We can **count** the number of functions $f : A \to B$ from one set A to another set B **without listing** them all.

To *refer* to them, *order* the elements of A, so can speak of first, second, etc. For example, suppose A has 4 elements and B has 7.

There are 7 possible outputs (in B) for the first input from A.

For each choice of output for first input, there are 7 possible outputs for the *second* input from A.

For each choice of outputs for first and second inputs, there are 7 possible outputs for the *third* input from A.

And for each choice of outputs for first, second, and third inputs, there are 7 choices for the output for the 4^{th} input.

Thus, altogether, there are

$$\underbrace{7 \times 7 \times 7 \times 7}_{4} = 7^4$$

functions from a 4-element set to an 7-element set.

We can **count** the number of **injective** functions $f : A \to B$ from one set to another **without listing** them all. Again suppose A has 4 elements and B has 7.

There are 7 possible outputs (in B) for the first input from A.

For each choice of output for first input, there are $7 - 1$ possible outputs for the *second* input from A, since the output for the second input must be *different* from the output for the *first* input.

For each choice of outputs for first and second inputs, there are $7 - 2$ possible outputs for the *third* input from A, since it must be different from *both* the first and second outputs (which are not the same as each other).

And for each choice of outputs for first. second, and third inputs, there are $7 - 3$ choices for the output for the 4^{th} input since it must different from the first three outputs (which are all different).

Thus, altogether, there are

$$7 \times (7 - 1) \times (7 - 2) \times (7 - 3)$$

injective functions from a 4-element set to an 7-element set.

Count the number of **orderings** of a 4-element set. (without listing them).

There are 4 choices for the first element of the subset.

For each choice of first element there are $4-1$ remaining choices for second element, since we can't re-use the first choice.

For each choice of first and second elements there are $4-2$ choices for third element, since we can't re-use the first two (different) choices.

And just $4-3$ choices for the last element. So, altogether,

$$4 \times (4-1) \times (4-2) \times (4-3)$$

orderings of a 4-element set.

The **factorial** function and notation is convenient: for non-negative integer n

$$n! = n(n-1)(n-2)\ldots 4 \cdot 3 \cdot 2 \cdot 1$$

By convention

$$0! = 1$$

So the number of orderings of a set with n elements is $n!$

The **binomial coefficients** are

$$\binom{n}{k} = \frac{n!}{k!\,(n-k)!} = n \text{ choose } k$$

Count the 3-element subsets of a 7-element set, (not listing).

There are 7 choices for the first element of the subset.

For each choice of first element there are $7 - 1$ choices for second element of the subset, since we can't re-use the first choice.

For each choice of first and second elements there are $7-2$ choices for third element of the subset, as we can't re-use the first two (different) choices.

But this approach imparts a fictitious **ordering** to the subset, and we must compensate.

We must **divide** by the number of possible orderings of 3 things, namely 3! from above.

Thus, the number of 3-element subsets of a 7-element set is

$$\frac{7(7-1)(7-2)}{3!} = \frac{7 \cdot 6 \cdot 5}{3!}$$

$$= \frac{7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1}{3! \cdot 4 \cdot 3 \cdot 2 \cdot 1} = \binom{7}{3}$$

Count the number of *surjective* functions $f : \{1, 2, 3\} \to \{4, 5\}$. Look at *all* functions to see *how* a function might fail to be surjective:

case 1:	$f(1)=4$	$f(2)=4$	$f(3)=4$
case 2:	$f(1)=4$	$f(2)=4$	$f(3)=5$
case 3:	$f(1)=4$	$f(2)=5$	$f(3)=4$
case 4:	$f(1)=4$	$f(2)=5$	$f(3)=5$
case 5:	$f(1)=5$	$f(2)=4$	$f(3)=4$
case 6:	$f(1)=5$	$f(2)=4$	$f(3)=5$
case 7:	$f(1)=5$	$f(2)=5$	$f(3)=4$
case 8:	$f(1)=5$	$f(2)=5$	$f(3)=5$

To miss one or the other target element (cases 1, 8) all inputs go to a *single* output. There are two choices of this. functions so the number of surjections is

$$\text{(no. all) - (no. failures)} = 2^3 - 2$$

To count surjections

$$f : \{1, 2, 3, 4, 5\} \to \{8, 9\}$$

again the only failures are functions which have a single output for all inputs, since the target set has just two elements.

The number of surjections is thus

$$\text{(no. all) - (no. failures)} = 2^5 - 2$$

since (as above) the number of *all* functions is 2^5.

To count surjections

$$f : \{1, 2, 3, 4, 5\} \to \{6, 7, 8, 9\}$$

take a different approach.

Since the target set has just one fewer than the source set, a surjective function can send just two inputs can be sent to the same output, and all others must go to different outputs.

So we count the number of 2-element subsets of $\{1, 2, 3, 4, 5\}$, and for each such choice there are $4(4-1)(4-2)(4-3)$ choices of outputs.

Thus, the number of surjections from 5-element to 4-element set is

$$\text{(no. 2-element subsets of source)} \times 4!$$

$$= \binom{5}{2} \times 4!$$