Cryptography

Design Project Proposal

by Cole Henrich

Classical Ciphers

CAESAR CIPHER

Encryption with any shift, specified by user.

Automatic Decryption,
via brute force, that is, generating a key for each possible unique shift, then using a proprietary `not_english()` class (which I have already written and which can tell non-english from English, be that non-english ciphertext or French. It does this by checking whether a certain proportion (as of this writing, 50%) or more of the words used in the text can be found in a quite exhaustive list of English words. I created this list by Scanning and retrieving the unique words out of a compendium of:

1. Part of Chapter 3 of "When the Emperor was Divine" by Julie Otsuka
2. The Entirety of "Great Expectations" by Charles Dickens
3. The Entirety of "War and Peace", by Leo Tolstoy, translated into English by Louise and Aylmer Maude
4. A Portion of "Dreams From my Father" by Barack Obama, at least the preface, introduction, and the first chapter.
5. The Entirety of "The Jungle Book" by Rudyard Kipling
6. The Entirety of "The Great Gatsby" by F. Scott Fitzgerald.

While I compiled this list using Scanner and its associates, I as of this writing have the lists compiled in 3 separate static arrays (static in terms of access and in terms of their mutability), which are split into 3 to avoid overflowing the 64 kb. limit on Java classes. These are then accessible through a BiggestArray Class which can return an array of the (however many, if I scan more books) arrays.
Update: I am, as of this second writing of the proposal, working on a class which will compile these array-storing classes when run (i.e., when there are new sources of text), so that the text-input goes straight into the static arrays. I consider it a "dormant compiler", because it compiles other classes, and then goes dormant (not modifying anything) until a new text source is input and it is called to run. This way, it strikes a compromise between Scanner recreating the list every runtime (bad) and the static arrays needing to be manually updated when the trainer-text is updated (bad), and the result is good.

I intentionally created the static (non-mutable) arrays to speed up processing times.

SUBSTITUTION CIPHER

Encryption using:

Auto-generated random key;

Pre-generated sets of substitution keys,
for those who want to play around; I created (not in connection with this project) a set of substitution encrypters and decrypters (which need the key, far different from the cipher cracker I'm making here); the set includes (as of this writing) substitution using emojis, substitution using permutations of the characters < and > which results in a cipher looking like a snake; various variations on this previous one, such as with permutations of commas and periods - graphically minimalist.
The set also includes a substitution cipher using most all of the keys on the qwerty keyboard, in general substituting alphabetic characters for alphabetic characters, but also including nonalphabetic characters out of the necessity to find encrypted equivalents for non-alphabetic characters like ! and ?.

Encryption using User Generated Keys.
Will be able to take at least a key for all alphabetic characters, I may extend that to more or even as many chars as they want. It can then encrypt and decrypt using their key.

VIGENÈRE CIPHER

Be able to crack a given vigenère cipher, and also create one (I will have to research this).

ENIGMA MACHINE

RSA ENCRYPTION

Official Proposal

I am trying to create an application which can crack and create classical ciphers, simulate modern cryptography like RSA encryption, and provide a user interface to encourage user experimentation with ciphers, e.g., a simulation of an enigma machine (yikes! If I can do it...)

Prerequisite Knowledge

I need to understand:

How RSA works
How the Enigma Machine works.
The theory of sharing keys, as used in RSA and modern cryptography.
The specifics of the Vigenère cipher.

A list of goals and subgoals for the project in its entirety

Classical Ciphers

Caesar

AutoCrack
AutoEncrypt
User-Specified Encryption
A Caesar generator, if the user wants to manually solve one for mental exercise.
Machine-Decrypt
I say this word because it is different from AutoCrack. AutoCrack, for any cipher here, means (or the act of) cracking a cipher solely given the enciphered text. Machine-Decrypt is when the user already knows the key, and they just want the computer to blaze through their text and translate each letter. I call it Machine-Decrypt because the functionality in use is not any of the cryptography theory generally in use elsewhere, but simply applying the ability of the computer (the machine) to do repetitive tasks.

Substitution

AutoCrack, not by brute force, but by a very (very very very) code-heavy approach, which tries to infer based on characteristic such as frequency, length, and repetition of letters and their placement within the words which letters are which, and grows from there - the more it knows, the better inferences it can make. As it assumes the identities of more letters (it is very possible for a surprise word which has the attributes of the presumed other word to throw the entire process off; this is why I say "assumes"), it can make "better" (if its assumptions are right) identifcations of words. Specifically, the more letters it assumes, the longer of words it can attack, and it can identify words based on 4, 5, or even 6 letters that it has already presumed identified, and then infer the last unknown. For example, "kitchen", if it already assumes, "itchen", then the first letter of that encrypted word is almost certainly k.

AutoEncrypt (autogenerated keys)

User Specified Encryption - elaboration can be found under initial overview of "Classical Ciphers" stated before the "Official Proposal".

Machine-Decrypt - see "Caesar" under "goals"

AutoEncrypt without giving the user the key, for mental exercise.
Although...user beware!

Vigenère

AutoCrack, Machine-Decrypt, AutoGenerate and give the user the key, AutoGenerate and don't give the user the key (poor user who asks for that...), generate given key.

I should make definitions for all these:

AutoCrack and Machine-Decrypt are already defined;
AutoGenerate/AutoEncrypt is common knowledge; its two subsets are "give the user the key" and "don't give the user the key";
and then there is *Generate With Key*, which is the counterpart of Machine-Decrypt - both function with known keys, whereas AutoCrack and AutoGenerate/AutoEncrypt (both subsets) function with the key unknown; in fact their whole purpose is to derive the key.

Back to Vigenère, I will have to research what is entailed by AutoCrack, AutoEncrypt, Generate With Key, including what it means to generate given a key - what does a Vigenère key look like?- and Machine-Decrypt - again, what is a Vigenère key like?, which informs how the program will process a given key. Whereas the Caesar and Substitution ciphers will have paltry graphics, the Vigenère, having more parts, may include more **graphics**, which may interest the user more and get them to think about the inner workings of the cryptographical theory that is deciphering their input. The same goes for the Enigma Machine.

ENIGMA MACHINE

I intend to create a interactive representation of the enigma machine, so that the user can play with the "gears" and alter its function "mechanically". I expect to give it as many interactive parts as it needs to create a real enigma cipher; if there are any non-essential parts (there may not be), I will not include those in the digital simulation for the sake of time. I hope to create both an enigma-maker and an enigma-cracker; if I have to choose between one in the case of a time crunch, I think I might choose the enigma-maker, since you can only create an enigma with an enigma machine, but maybe you can crack it another way. Still, I intend to do both if I can. There is a nice enigma machine on cryptii.com, but I intend to make mine a little more like a "machine", with at least a few graphical representations - e.g., an image of a gear that you can turn, maybe.

RSA ENCRYPTION

I don't know enough about it yet to say exactly my goals. I'll try to mirror it as faithfully as possible, but with smaller numbers.
I think I will also have explanations, written, of all the cryptographical theories and ciphers that I am using in the project, in case the user is interested.

Design Limitations

I expect that there will be some functionality missing from the enigma machine in terms of simulating all of its mechanical parts; I may not be able to send the private key for RSA through the program (users may have to rely on just texting or emailing, but maybe, in some long shot, I might give it that ability).

I also know that in the current version of my substitution cipher, there is this flaw: it starts cracking the cipher by assuming that the first four letter word, which starts and ends with the same letters, that it encounters, is the enciphered equivalent of the word "that". This is most often correct, since "that" is one of the most common four letter words, hence it is most likely to occur before the others, and as long as it comes first (in this algorithm) it's ok. What matters most, though, is "that" is by far the most common 4 letter word which starts and ends with the same letter. For example, if I see the text "qreq" very frequently throughout a substitution cipher, qreq most likely equals "that", and hence I can add q = t, r = h, e = a to the key I am compiling. This is exactly what the algorithm does. Having assumed h, a, and t, it moves on the other inferences using those assumptions. I won't go in depth here.

So after all that pretext, here (finally) is the flaw: what if qreq is *not* "that"? Answer: the cipher **will not** be cracked. Not with this algorithm, on that piece of text. As long as I can, I will methodically minimize the chances of such mistakes occuring. However, they will never be completely eradicated, as long as I am using algorithms which run by assuming the identity of words based on the composition of their letters.

Finally, the RSA functionality will not be able to use prime numbers as long as are in use for actually encrypting people's data.