Crypto: A Cryptogram Encoder/Decoder User's Guide

1 Introduction

Crypto is a Linux/Unix/Mac OS X program for encoding and decoding cryptograms. There are two main versions of the program, in which the decoding process is structured as a game: one with a terminal-based user-interface, and one with an X window system graphical user-interface. But there is also a version of the program with an automatic (batch) user-interface.

2 Usage of Terminal and Graphical Versions

This section describes how the two main versions of the program can be used. A later section describes the usage of the version with the automatic user-interface.

Crypto makes use of a *lexicon*, *lex*, consisting of a set of *words*, which are nonempty sequences of lowercase letters. Initially, crypto's lexicon is empty.

Crypto has primary and secondary command loops. In the terminal-based version of crypto, commands may be abbreviated to unambiguous prefixes of themselves, and a command's arguments are listed after the command, separated from themselves and the command by whitespace characters. In the graphical version of the program, commands are selected by clicking on buttons, after which the user is prompted to enter or select any command arguments.

Upon invocation, crypto enters its *primary command loop.* The primary commands are listed below. In the terminal-based version, there is also a help command.

- The quit command causes crypto to exit with status success (0). In the terminal-based version of crypto, signaling end-of-file also has this effect. In the graphical version, asking the window manager to request that the program exit will also have this effect.
- The lexicon command takes an argument, *file*, and replaces the lexicon, *lex*, with the words of *file*. Words are separated by nonempty sequences of whitespace characters. If the file cannot be opened for input, then crypto issues an error message and leaves *lex* unchanged. In the graphical version, the user is given the option of editing the filename and trying again. If non-words appear in *file*, then warning messages are issued, but those non-words are otherwise ignored.
- To understand what the encode command does, we need some definitions.

  A *message* is a list of *lines*, each of which is a list of words. A *renaming ren* is a function from lowercase letters to lowercase letters with the property that, for all lowercase letters y, there is a unique lowercase letter x such that $ren(x) = y$. We *apply* a renaming *ren* to a message by applying *ren* to each letter of each word of each line of the message. A *decoding* of a message *msg* is a message msg' such that

- each word of msg' is in the lexicon, *lex*;
- msg' can be formed by applying some renaming to *msg*.

The encode command begins by asking the user to enter a message. The words of the message may be spread over multiple lines, and words are separated by whitespace characters. In the terminal-based version, the message is terminated by typing a line consisting of the character "." (followed by a newline); input of the message may be aborted by typing a line consisting of the character "!". In the graphical version, after typing the message in a text window, the user has the option of confirming hir[1] choice of message by clicking the OK button, or aborting the encode command by clicking the Cancel button. If any non-words were entered by the user, then encode displays an error message. In the graphical version, the user is given the option of editing the message and trying again, but the terminal version simply aborts.

Let the message *msg* be the list of words that the user entered. Next, encode checks whether *msg* is the only decoding of itself. The user is allowed to abort this computation, by typing hir interrupt character (usually *CTRL*-c) in the terminal-based version, and clicking the Cancel button in the graphical version. During the computation, the user is periodically informed of its progress.

If *msg* is not the only decoding of itself, then an error message is issued. In the graphical version, the user is given the option of editing the message and trying again; the terminal version simply aborts. Otherwise, a random renaming is generated and then applied to *msg*, and the resulting encoded message is displayed (*msg* will be the only decoding of this message).

- The decode command begins by reading a message *msg* from the standard input, as with the encode command. Next, decode checks whether *msg* has a unique decoding (and saves this decoding for future use, without telling the user what it is, if the answer is "yes"). As with the encode command, the user is allowed to abort this process.

  If *msg* doesn't have a unique decoding, then an error message is issued. In the graphical version, the user is given the option of editing the message and trying again; the terminal version simply aborts. Otherwise, crypto enters its *secondary command loop*. At each iteration of the secondary command loop, crypto first displays the

[1] "Sie" (pronounced "see", replacing he/she) and "hir" (pronouced "hear", replacing him/her) are gender neutral pronouns.

current *partially decoded message*, *pdm*, consisting of a sequence of *partially decoded lines*, each of which consists of a sequence of *partially decoded words*, i.e., nonempty sequences of upper- and lowercase letters. Initially, this partially decoded message is *msg*. The uppercase letters represent the renamings already performed by the user and program.

It will always be the case that the current partially decoded message, *pdm*, is *consistent* with the message, *msg*, being decoded, i.e.,

- *msg* and *pdm* have the same shape, and
- for all lowercase letters a, either
  - a appears in *msg* at exactly the same positions as it appears in *pdm*, or
  - there is an uppercase letter b such that a appears in *msg* at exactly the same positions that b appears in *pdm*.

Suppose *pdm* is a partially decoded message that is consistent with the message, *msg*, being decoded, and that *msg*′ is the decoding of *msg*. We say that *pdm* is *decodable* iff each occurrence of an uppercase letter in *pdm* appears in the same position in *msg*′, but in its lowercase form. We say that *pdm* is *decoded* iff *pdm* is decodable and has no lowercase letters. Given a lowercase letter a that appears in *pdm*, we say that the *decoding* of a is the lowercase letter b that appears in *msg*′ at the positions corresponding to the positions at which a appears in *pdm* (i.e., the positions at which a appears in *msg*).

The secondary commands are listed below. In the terminal-based version, there is also a `help` command.

- The `quit` command causes `crypto` to exit with status success (0). Signaling end-of-file (normally *CTRL*-`d`) in the terminal-based version also has this effect. In the graphical version, asking the window manager to request that the program exit will also have this effect.
- The `abort` command causes `crypto` to return to its primary command loop.
- The `check` command works as follows. If the current partially decoded message, *pdm*, is not decodable, then the user is informed of this fact. Otherwise, if *pdm* is decoded, then the user is informed of this fact, and `crypto` returns to the primary command loop. Otherwise, the user is told that *pdm* is decodable, but isn't yet decoded.
- The `hint` command works as follows. If the current partially decoded message, *pdm*, is not decodable, then the user is informed of this fact, and no other action is taken.

Otherwise, if *pdm* is decoded, then the user is informed of this fact, and crypto returns to the primary command loop.

Otherwise, crypto selects the lowercase letter *a* of *pdm* that appears at least as many times in *pdm* as any other lowercase letter, and that appears earlier in the alphabet than any other lowercase letter with an equal number of occurrences in *pdm*. Crypto then produces a new partially decoded message *pdm*′, by replacing all occurrences of *a* in *pdm* by the uppercase version of the decoding of *a*. The current partially decoded message is then set to *pdm*′.

– The replace command has two arguments *a* and *b*, where *a* is a lowercase letter of *pdm*, and *b* is a lowercase letter than doesn't appear in uppercase form in *pdm*. It then produces a new partially decoded message *pdm*′, by replacing all occurrences of *a* in *pdm* by the uppercase form of *b*. The current partially decoded message is then set to *pdm*′.

– The undo command causes crypto to undo the most recent replacement made, or to return to the primary command loop, if there is no such replacement. A given replacement might have been carried out by the user using a replace command, or might have been carried out by the program using a hint command.

3 Options and Customization

The terminal-based version of crypto takes no command-line arguments. If it is given any arguments, it prints a usage message and exits with status failure (1).

The graphical version of crypto can be controlled and customized using command-line arguments and X resources.

The *name* (a string of letters, digits, `-`'s and _'s) and *display* of the program can be customized using command line arguments:

```
-name name
-display display
```

If these aren't used, then *display* is taken from the value of the `DISPLAY` variable in the user environment (`:0.0` if it isn't set), and *name* is the last part (everything past the last `/`, if any) of the name by which the program was invoked (i.e., `crypto`, unless the shell script is renamed).

Some program attributes may be customized using the X resource database:

Resource Name	Default	Comment
`background`	`white`	background color of windows
`foreground`	`black`	foreground color of windows
`font`	`10x20`	font of windows

When looking for the value of a resource attribute *attr*, crypto looks for the value of the resource name

name.attr

For instance, a user could load any of the following resource specifications into the X server's resource database to set crypto's background color to `grey`:

```
*background: grey
name*background: grey
name.background: grey
```

(Later forms take precedence over earlier ones.)

These attributes may also be specified using command line arguments, which take precedence over the settings in the X resource database:

Option	Attribute
`-background` *color*	background color
`-bg` *color*	background color
`-foreground` *color*	foreground color
`-fg` *color*	foreground color
`-font` *font*	font
`-fn` *font*	font

Later command line arguments take precedence over earlier ones, and command line arguments may be abbreviated to unambiguous prefixes.

If invalid command line arguments are supplied to crypto, or if crypto is unable to open the display, then it issues an error message on the standard error output and exits with status failure (1).

4 Automatic User-Interface

The automatic user-interface version of crypto is invoked with two arguments: a lexicon filename, followed by a mode, which may be encode or decode. Modes may be abbreviated to unambiguous prefixes. If the arguments are invalid, or if any other kind of error occurs, this version of crypto exits with status failure (1), after printing an appropriate error message on the standard error output. Warning and status messages are also printed on the standard error output. At any point, the user may terminate crypto by typing hir interrupt character (usually *CTRL*-c).

Crypto begins by loading its lexicon from the lexicon file, issuing warning messages about any non-words appearing in this file.

If the mode is encode, then crypto reads a message from the standard input, where the message is terminated by end-of-file. If the message contains any non-words, then an appropriate error message is issued, and crypto exits. Otherwise, crypto checks whether the inputted message is the only decoding of itself. The user is periodically informed of the status of this checking process. If the inputted message is not the only decoding of itself, then an error message is issued, and crypto exits. Otherwise, a random renaming is generated and then applied to the message, the resulting encoded message is printed on the standard output, and crypto exits with status success (0).

If the mode is decode, then crypto reads a message from the standard input, where the message is terminated by end-of-file. If the message contains any non-words, then an appropriate error message is issues, and crypto exits. Otherwise, crypto checks whether the inputted message has a unique decoding, saving this unique decoding if the answer is "yes". The user is periodically informed of the status of this checking process. If the inputted message doesn't have a unique decoding, then an error message is issued, and crypto exits. Otherwise, the unique decoding is printed on the standard output, and crypto exits with status success.