353 — Computer Security — Simple Block Encryption

Project (Get Acquainted Phase) – Simple Block Encryption
Introduction

This project is to build a visualization of a very simple block encryption algorithm. The block encrypter will take as input a plaintext message and a password, and it will encrypt each block of the text the password, and display the "prepared" plaintext message in blocks on a grid (one character per grid cell, with a gutter between blocks) and also display each cyphertext block underneath its plaintext block.

The prepared plaintext blocks will include a sequence number (a single digit) at the front of the block, so that on decryption, their order in the plaintext message can be confirmed.

The password should be verified as following the complex password policy that is called "comprehensive8, and be a fixed 8 characters long (although in the real world, in comprehensive8 8 chars is a only lower limit). Note that the password will be the same as the block size.

We will assume that the plaintext message is simple printable 8-bit ascii characters (plus the space character, ascii 32), so no non-printing characters should be allowed.

The plaintext message will be at most 27 chars long, because we want it to fit easily on a screen line, and also because each block will need a 1 character sequence number (in 1..4). And in addition, we will add a char-count-ish final character which describes the actual length of the user's plaintext message. Thus, this final character, plus the 4 block numbers (one character each), plus the 27 character message (padded as needed) will make up 32 characters in all -- 4 blocks of 8 characters each.

For the encryption algorithm, we will use the ECB mechanism. First, if the plaintext message is less than 27 characters, pad it out with spaces. Then append a 28th character that represents the actual length of the user's plaintext message as follows: add 32 to the actual length to get the (printable) ascii character code to use.

Next, create the 4 plaintext blocks, each with a leading sequence digit (1 through 4). This gives four 8-char blocks. We will assume the plaintext block characters are all in the range 32 to 127 (i.e., in 7-bit ascii). We can encrypt a single plaintext char, A, with a single password char, P, by doing the following function:

Encode
o1- Let ax = A - 32; px = P - 32;
o2- Let bx = Logical-OR of ax and px
o3- Let cx = bx + 32;

cx is the cyphertext block char for this simple XOR mechanism. This XOR mechanism is invertible by running it a second time. That is, the Decode function is the same as the Encode function -- just feed in the cyphertext block char and the same password char and out pops the original plaintext block char.

Each plaintext block char will be matched up to its corresponding password char and then converted this way to its cyphertext block char.

Display

The program will be written in **P5.js+Javascript** with an **HTML** web page for display (as described in lecture). You may use all, some or none of the example running code provided. You do not have to show grid lines or row/column numbers.

Your program will include an input text box for the user's plaintext, and an input text box for the user's password. And a button for the user to press to "submit" the plaintext and password to your encryption program, which will then display the results in a grid.

Team

Your team may contain up to four members. We would prefer 3-4 sized teams. Pick a short-ish (<= 12 letter and/or number) name for your team (e.g., "ABX", or "Groggy", or "we-like-cats", etc.). If you are on your own, pick a 1-person team name. If you want to join or form a team but can't (?), tell me at start of class and I will hold an auction for you.

Naming Docs

Name your documents/files like this: "`353-05-p1-Groggy-<type>-<yymmdd>.<ext>`" where <ext> is "pdf", "zip", etc., and where <type> is "Standup", or missing for the project zip file; and where `<yymmdd>` is for example 210903 for Sept 3rd.

Project Development Reporting

Standup Status Report, twice weekly. The Standup Status Report is due **Monday's** and **Friday's** by noon-ish. One report per team, **CC'ing the other team members**. It should contain, team name and the member names, **but no CWIDs** (which ought to be private). This document should be delivered **as a PDF file**; and the **filename** should be in the following format: include your course and section number, project number, your team name, the document type (Standup), and the date as YYMMDD:. E.g., "`353-05-p1-Groggy-Standup-210231.pdf`".

Standup Status Report contents should be, for each team member, a list of the 3 **Standup question short answers**: Q1: what have you **completed** (name sub-tasks) by the time of this Standup report; Q2: what do you **plan to complete** (name sub-tasks) by the time of the next Standup report; and Q3: what obstacles if any (1-line description) are **currently blocking you** (for which you've reasonably tried to find the answers by yourself, including asking your team about them – known as "due diligence"). Note, that you can email the professor, or ask questions during office hours to get answers.

Readme File

When your project is complete, you should provide a README.txt text file. Be clear in your instruction on how to build and use the project by providing instructions a novice programmer would understand. If there are any external dependencies for building, the README must also list them and how to find and incorporate them. Usage should include an example invocation. A README would cover the following:

- Class number
- Project number and name
- Team name and members
- Intro (including the algorithm used)
- Contents: Files in the .zip submission
- External Requirements (None?)
- Setup and Installation (if any)
- Sample invocation
- Features (both included and missing)
- Bugs (if any)

Academic Rules

Correctly and properly attribute all third party material and references, lest points be taken off.

Submission

All Necessary Files: Your submission must, at a minimum, include a plain ASCII text file called **`README.txt`**, all project documentation files (except those already delivered), all necessary source files to allow the submission to be built and run independently by the instructor. [For this project, no unusual files are expected.] Note, the instructor not use use your IDE or O.S.

Headers: All source code files must include a comment header identifying the author, author's contact info (please, no phone numbers), and a brief description of the file.

No Binaries: Do not include any IDE-specific files, object files, binary **executables**, or other superfluous files.

Project Folder: Place your submission files in a **folder named** like your Standup report files: `353-05-p1-Groggy`.

Project Zip File: Then zip up this folder. Name the .zip file the **same as the folder name**, like `353-05-p1-Groggy.zip`. Turn in by 11pm on the due date (as specified in the bulletin-board post) by **submitted via emailed zip file(s) (preferred)**, or via accessible cloud (eg, Github, Gdrive, Dropbox) with emailing the accessible cloud link/URL. See the Syllabus for the correct email address. The email subject title should include **the folder name**, like `353-05-p1-Groggy`. Note that some email-programs block .ZIP files (but maybe allow you to change .ZIP to .ZAP) and block sending zipped files with .JS files inside (but maybe allow you to change .JS to .JS.TXT).

Email Body: Please include your team members' names (but not your IDs) at the end of the email.

Project Problems: If there is a problem with your project as submitted, don't put it in the email body – put it in the README.txt file, under an "ISSUES" section. Don't forget, I have office hours as well.

Grading

- 80% for compiling and executing with no errors or warnings
- 10% for clean and well-documented code (Rule #5(Clean))
- 5% for a clean and reasonable documentation files
- 5% for successfully following Submission rules