GROUP: SPENCER STEVENS, CHINMAY LOHANI, XINYUE HUANG

# OFFLINE CAPTCHA FOR FILE ENCRYPTION

## CAPTCHA

### Completely Automated Public Turing test to tell Computers and Humans Apart

Offline CAPTCHAs are a type of CAPTCHA system that works without needing an internet connection. These are designed to generate deterministic puzzles and validate the “human user” based on the puzzle solutions. This makes them suitable for applications that need to function in environments where internet connectivity is unreliable or not available, including encryption and decryption of files. CAPTCHAs can be used as an additional layer of security when encrypting or decrypting files, particularly in offline environments, and involves generating a CAPTCHA challenge that must be solved correctly before a file can be encrypted or decrypted.

### Why CAPTCHAs?

CAPTCHAs, including offline ones, serve as gatekeepers, filtering out automated threats while allowing genuine human users to proceed. They play an indispensable role in protecting platforms from many potential threats and abuses, including:

1. Prevent Unauthorized Access: CAPTCHAs can help prevent unauthorized users from accessing, encrypting, or decrypting files. This is particularly useful in scenarios where files are stored in shared or public locations.
2. Mitigate Brute-Force Attacks: By requiring a CAPTCHA challenge to be solved before a file can be encrypted or decrypted, the system can effectively mitigate brute-force attacks. This is because each failed attempt at solving the CAPTCHA results in a delay, making brute-force attacks time-consuming and impractical.
3. Enhance Offline Security: In offline environments where internet connectivity is unavailable or unreliable, offline CAPTCHAs provide an additional layer of security which is reliable even in the case of hardware compromise.

### Challenges

1. Usability Issues: CAPTCHAs, particularly those that are complex or difficult to solve, can lead to usability issues, where users may become frustrated if they are unable to solve the CAPTCHA, potentially leading them to abandon the task.
2. Accessibility Concerns: CAPTCHAs can pose challenges for individuals with visual impairments or cognitive disabilities, which if not designed with accessibility in mind, these users may be unable to encrypt or decrypt files.

3. Limited Security: While CAPTCHAs can deter casual attackers, they may not be sufficient to stop determined attackers with advanced tools, e.g., sophisticated bots may be able to bypass CAPTCHA challenges, potentially compromising the security of the files.

## Our solution

Considering the mentioned challenges with offline CAPTCHAs for file encryption/decryption, we present 3 potential solutions to the CAPTCHA puzzles that overcome these limitations while maintaining the existing security.

### Sudoku puzzle

Sudoku puzzle is a popular number placement game that requires players to fill out a 9 X 9 grid so that each row, each column, and each 3 X 3 sub-grid contain numbers from 1 to 9. It's a logical puzzle that needs players to try every different and possible placement to achieve the final solution, so the rule and logic behind this game can effectively test whether the player is a human. It would be too time-consuming for a computer to OCR the randomly generated puzzle picture and solve it each time to decrypt.

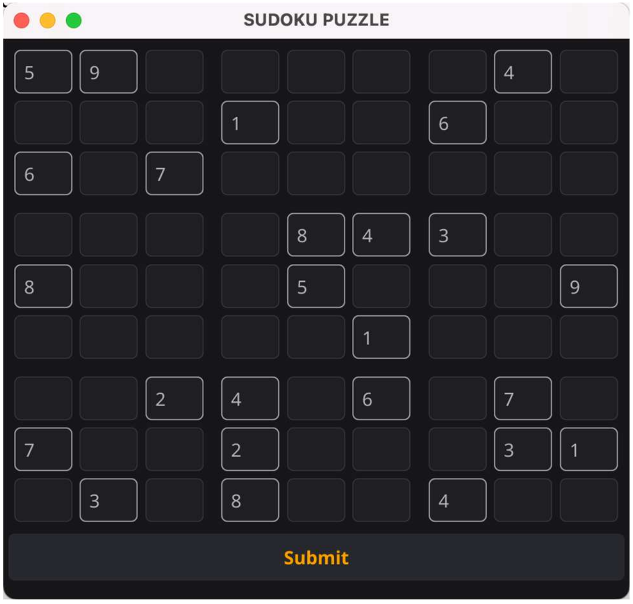
The implementation of sudoku puzzle is that the final key which is used for encryption and decryption is split into two parts. One part is the hashed value of the user's shared secret, and the other part is the hashed value of the solution of the sudoku puzzle. Then we combine two parts. The application did the following steps.
First, application waits for the user to input shared secret.
Next, it uses the shared secret as seed to generate a sudoku puzzle.
Then, it accepts user's solution to the puzzle and compares it with the correct solution.
Finally, if the user solution is correct, the application will execute user-specified functions such as encryption or decryption. If it fails, it will terminate.

The first advantage of this puzzle is that it is **costly** for a computer to brute force. Our sudoku puzzle is **randomly generated** based on the hash value of the user's weak credential. Hence, each time sudoku puzzle is randomly generated instead of reading a fixed puzzle from a large library, which can effectively prevent attackers from brute-forcing the sudoku library offline without the logical restrictions of our software. Also, it makes the size of our captcha encryption application much lighter because the puzzle is generated based on the input user key without a database included. Moreover, the generation takes time, which costs time and resources to break.

The second advantage of the sudoku puzzle is **determinism**, which means that the final key is one-way and unique. Real human users only need to solve the same puzzle once during encryption or decryption while for computers to brute force, they would have to test every credential in the dictionary and solve all puzzles to get the final key. To achieve determinism, our generator accepts user input, which is a shared secret, as a **seed** to generate the puzzle to ensure it is reproducible. Also, during puzzle generation, we checked the number of puzzle solutions in puzzle generation to make sure there was only **one solution**. In this way, the final key will be unique.

The third advantage of the sudoku puzzle in this project is that it can serve as a “**human check**” in the application. The generated sudoku puzzle is **human-solvable** but also ensures the difficulty for solving. It can be used to check whether the user is human or computer.

The fourth advantage of sudoku in this application is a nice **GUI**. We introduced **fyne.io** library to make it more user-friendly instead of input value in the terminal. The interface will show results when click on submit to verify our puzzle solution.



| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | 9 | | | | | | 4 | |
| | | | 1 | | | 6 | | |
| 6 | | 7 | | | | | | |
| | | | | 8 | 4 | 3 | | |
| 8 | | | | 5 | | | | 9 |
| | | | | | 1 | | | |
| | | 2 | 4 | | 6 | | 7 | |
| 7 | | | 2 | | | | 3 | 1 |
| | 3 | | 8 | | | 4 | | |

However, the sudoku puzzle is not completely unbreakable. It also has some flaws which make it not the best option for captcha offline encryption compared to chess puzzle, instead, it is better to serve as a “human check” to examine whether the input is from a human or computer.

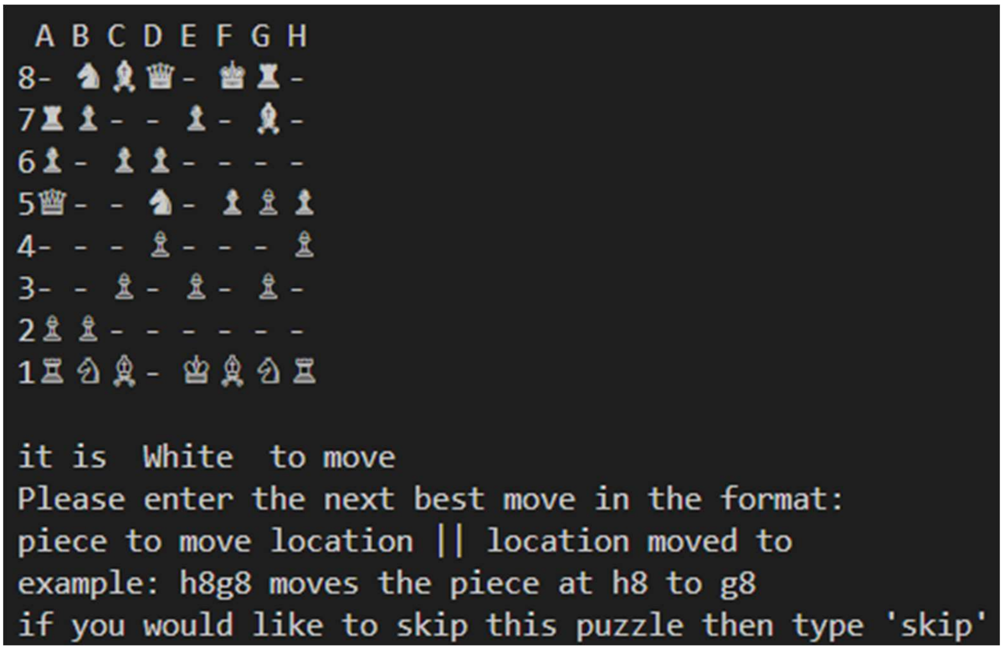
The first drawback is that the puzzle is **algorithm-solvable**. Although it would take a long time for brute forcing, with supercomputing and advanced algorithms, it still can not completely prevent attackers from dictionary attacks.

The second drawback is that attackers may use advanced techniques to **bypass** the puzzle part to achieve the final key. The generation of sudoku puzzles relies on the generator to fill out all the grids, which means we will generate the solution to the puzzle at the beginning. Then we choose some grids and leave them empty to form our final puzzle. The difficulty of the puzzle is also based on the number of blank grids. However, when we run the offline application, the solution is generated before the user enters their answer. Suppose attackers can do reverse engineering and look up in computer memory. In that case, they may be able to find the pre-generated solution to skip solving the puzzle and combine two keys to get the final key. The captcha will be useless.

## Chess Puzzles

These puzzles are intended to be difficult to calculate efficiently for a computer but be relatively easy for a human to do quickly. The intent is that two computer of similar capabilities will end up at the same solution given the state of a chess board. There are a few bugs but it mostly works.

The library finds large swings in evaluations on the board and labels them as "puzzle points". Then has the user calculate the best move in the position and uses that best move to form a puzzle key.



```
 A B C D E F G H
8- ♞ ♝ ♛ - ♚ ♜ -
7♜ ♟ - - ♟ - ♝ -
6♟ - ♟ ♟ - - - -
5♛ - - ♞ - ♟ ♙ ♟
4- - - ♙ - - - ♙
3- - ♙ - ♙ - ♙ -
2♙ ♙ - - - - - -
1♖ ♘ ♗ - ♔ ♗ ♘ ♖

it is  White  to move
Please enter the next best move in the format:
piece to move location || location moved to
example: h8g8 moves the piece at h8 to g8
if you would like to skip this puzzle then type 'skip'
```

Bugs
The evaluation of a position may fluctuate slightly which may cause positions close to the cutoff point be lost.

The difference in capabilities for computers can show up drastically when calculating "puzzle points"

## Cryptographic Hash Puzzles

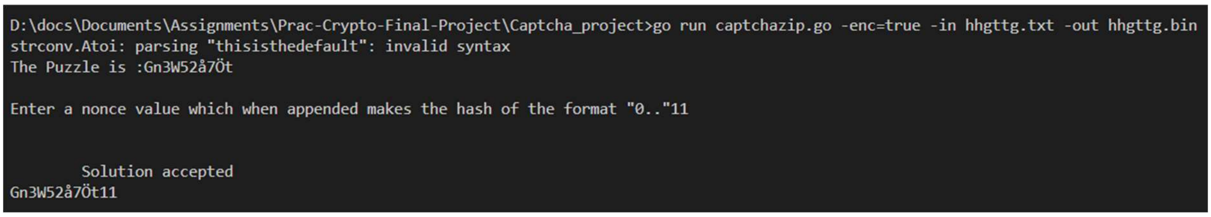
A cryptographic hash puzzle can be thought of as a complex puzzle with a unique feature: it can only be put together in one way. This puzzle, or hash, takes in data of any size and churns out a fixed-length string of characters, regardless of the size of the input. In the context of cryptocurrencies like Bitcoin, miners are constantly trying to solve these puzzles. In the context of file encryption and decryption with offline CAPTCHAs, the hash puzzle could be used as an additional layer of security and add the asymmetry in the process.

## Working Concept

Puzzle Generation: When a file is to be encrypted or decrypted, a cryptographic hash puzzle is generated. This involves getting an initial input ‘s’ from the user, and then generating the puzzle ‘x’, prompting user to find a number called a nonce, such that h(‘x’+nonce) == “000…”.

Puzzle Solution: The user (or the system) must solve the puzzle before the file can be encrypted or decrypted, where they are prompted to submit a solution that is valid {of format “000…”}, once accepted the system will generate a proof of its own that will be appended to the initial seed ‘s’ and return as the key value.

File Encryption/Decryption: Once the puzzle is solved, the file can be encrypted or decrypted. The key is generated by appending a system generated solution to the initial seed value. The proposed solution is an alternative and the nonce solution could be used as part of the encryption/decryption key (used in our implementation, or it could simply serve as a gatekeeper, preventing the encryption/decryption process from starting until the puzzle is solved, adding asymmetry).



```
D:\docs\Documents\Assignments\Prac-Crypto-Final-Project\Captcha_project>go run captchazip.go -enc=true -in hhgttg.txt -out hhgttg.bin
strconv.Atoi: parsing "thisisthedefault": invalid syntax
The Puzzle is :Gn3W52å7Öt

Enter a nonce value which when appended makes the hash of the format "0.."11


        Solution accepted
Gn3W52å7Öt11
```

## Scope of Improvement

The current condition for the hash puzzle solution to be accepted is very lenient making it easier for the user to get a solution, the process can be made more complex making it harder for computers with high processing power to achieve the solution in less time.