# CybAware

Beh Chuen Yang 4S201 (Leader)
Tan Guan Lin 4S122 (Member)

# Introduction

## Rationale

Singapore has a problem with password security. Recent attacks performed on SingHealth and MOE have extracted all sorts of sensitive information from as many as 1.5 million patients' details, including Prime Minister Lee Hsien Loong's personal health data, and 50,000 government staff account credentials, which were found to be sold online, solely through a select few users with bad online habits such as using simple passwords or repeating passwords over numerous platforms.

Despite this, in a survey done by the Cybersecurity Agency of Singapore, 50%~70% of respondents either do not change their password or only do so when prompted. An astonishing 60% adopted this exact habit even for their bank accounts, showing their complacency even when the risks of getting hacked have been proven again and again. Many commercial password generators (such as LastPass and Dashlane) are also susceptible to being intercepted either by social engineering or network poisoning, rendering the password useless while the user is completely oblivious.

## Objective

Our project, CybAware, aims to reduce the prevalence of cyber attacks related to passwords by:

1. Creating a password generator with a secure custom algorithm
2. Compare our generated passwords against those of commercial password generators with the help of web-crawling to demonstrate their strength
3. To save the aforementioned passwords safely in a password manager for convenience and security

## Literature Review

| |
|---|
| Citation of Article: **Bernstein, D. J. (2007).** ***The Salsa20 family of stream ciphers*** **(Unpublished master's thesis). The University of Illinois at Chicago. Retrieved April 20, 2019, from https://cr.yp.to/snuffle/salsafamily-20071225.pdf.** |
| This paper describes in great detail how the Salsa20 family of stream ciphers operate, both algorithmically and on the hardware level. The author, the developer of said algorithm, also goes into detail in establishing his cryptographic algorithm as a fast and secure symmetric cipher, citing many studies involving cryptanalysis of the Salsa20 algorithm, and critically analyses possible modifications to the algorithm, such as order of operations, the substitution of addition with multiplication and use of programming techniques. |
| Assumptions: The writer's tests are performed on his machine, with a unique processor architecture that optimizes performance for his variation of the algorithm. Security level and execution speed may decrease if used on typical architectures. |
| Implications of Author's Work: The development of the Salsa20 algorithm has created new possibilities in terms of cryptographic applications, for example a cryptographically secure pseudorandom number generator based on ChaCha20, a relative of the Salsa20 algorithm. |
| Impact of Author's Research: The author's research has given us a deeper understanding of the Salsa20 algorithm, and has given a great deal of flexibility within its implementation, allowing us to customize it to create our password generator. |

| |
|---|
| Citation of Article:**Leonhard, M. D. (2006, October 30). A Comparison of Three Random Password Generators. Retrieved May 16, 2019, from https://pdfs.semanticscholar.org/e6da/b609de895e1752e96691da9023ed1994b27e.pdf** |
| This paper analyses three password algorithms (AlphaNum, Diceware, Pronounce3) with various metrics, documenting their strength with the use of password entropy, their weaknesses through citing other papers and performing their own cryptanalysis, and the code implementation of the password algorithms. The writer also conducts a survey in which he takes into account user opinion, incidentally revealing that people have a very hard time at remembering passwords. |
| Assumptions: The writer's sample size for his survey is n=19, a figure too small to be accurate. He also assumes that Diceware users do not use expanded word lists, opting instead for the standard 7776-word list. |
| Implications: The author's analysis of these algorithms paves the way for more in-depth analysis of the algorithms, and is informative of when and how to use a password generation algorithm. |
| Impacts: This paper has assisted our project greatly in filtering out candidates for password analysis of commercial password generators and our own. We have selected LastPass (an implementation of AlphaNum featuring an extended character set) as an analysis candidate courtesy of this paper. |

| Citation of Article: **Ian Boyd (n.d.). Which hashing algorithm is best for uniqueness and speed? Retrieved from https://softwareengineering.stackexchange.com/questions/49550/which-hashing-algorithm-is-best-for-uniqueness-and-speed** |
|---|
| In this website, the author writes about various hash algorithms, including various metrics like collision ratio, execution times and randomness diagrams, to create comparisons between the quality and speed of hash generation. |
| Assumptions: Classification of algorithms based on security was done subjectively. |
| Implications:The author's analyses creates useful statistics for future authors to build on and analyse in comparison to other hashing algorithms. |
| Impact: The author's analyses have exposed us to MurmurHash2, and have spurred our considerations for application in the password generation algorithm. |

| Citation of Article: **VowpalWabbit. (n.d.). VowpalWabbit/vowpal_wabbit. Retrieved June 20, 2019, from https://github.com/VowpalWabbit/vowpal_wabbit/wiki/murmur2-vs-murmur3** |
|---|
| This website compares two iterations of the MurmurHash family of hash algorithms, MurmurHash2 and MurmurHash3. Through the use of various metrics and data analytics, the authors reach the conclusion that both algorithms have their pros and cons, with MurmurHash3 gaining the edge. |
| Assumptions: The authors are working only with the data set provided by the author of the algorithm. |
| Implications: The author's analysis creates useful statistics for future authors to build on and analyse in comparison to other hashing algorithms. |
| Impacts: This paper has informed us on the advantages of MurmurHash3 over MurmurHash2 and has provided us with convincing data to adopt the MurmurHash3 algorithm. |

| |
|---|
| Citation of Article: **Issue 552749 - chromium - An open-source project to help move the web forward. - Monorail. (n.d.). Retrieved from https://bugs.chromium.org/p/chromium/issues/detail?id=552749** |
| This error thread documents that the Javascript crypto library uses a weak cryptographically-secure pseudorandom number generator, through exposing a weakness in the stream cipher used. (RC4 displays bias in the sample distribution) |
| Assumptions: The user is on a Chrome web browser. |
| Implications: The getrandomvalues function under the crypto library in javascript is no longer secure |
| Impact: This has made us aware of security flaws within our current algorithm, and prompted us to search for alternatives to secure our random generation pathway. |

| |
|---|
| Citation of Article: **Silver, D., Jana, S., Chen, E., Jackson, C., & Boneh, D. (n.d.). Password Managers: Attacks and Defenses. Retrieved from https://crypto.stanford.edu/~dabo/pubs/papers/pwdmgrBrowser.pdf** |
| This study studies the security of popular password managers and their policies on automatically filling in Web passwords. It examines browser built-in password managers, mobile password managers, and 3rd party managers. It observes significant differences in autofill policies among password managers. Several autofill policies can lead to disastrous consequences where a remote network attacker can extract multiple passwords from the user's password manager without any interaction with the user. It experiments with these attacks and with techniques to enhance the security of password managers. It shows that their enhancements can be adopted by existing managers. |
| Assumptions: The study assumes the password manager is implemented in a Web platform. |
| Implications: This shows that most password generators are not secure, given that a simple malicious website is able to steal passwords with ease. |
| Impact: This has made us more aware of the demand of a secure password generator due to the relative inaction of said password generator companies after the publishing of this study, and the areas of improvement in creating password generators and managers. |

| |
|---|
| Citation of Article:**Stark, E., Boneh, D., & Hamburg, M. (n.d.). Symmetric Cryptography in Javascript. Retrieved from https://crypto.stanford.edu/sjcl/acsac.pdf** |
| This article takes a systematic approach to developing a symmetric cryptography library in Javascript. It studies various strategies for optimizing the code for the Javascript interpreter, and observe that traditional crypto optimization techniques do not apply when implemented in Javascript. It proposes a number of optimizations that reduce both running time and code size. Its optimized library is about four times faster and 12% smaller than the fastest and smallest existing symmetric Javascript encryption libraries. In addition, it shows that certain symmetric systems that are faster than AES when implemented in native x86 code, are in fact much slower than AES when implemented in Javascript. As a result, the choice of ciphers for a Javascript crypto library may be substantially different from the choice of ciphers when implementing crypto natively. Finally, it studies the problem of generating strong randomness in Javascript and give extensive measurements validating our techniques. |
| Assumptions: Precomputing techniques are used whose results may vary from computer to computer. |
| Implications: A cryptographic library has been developed for the Javascript community. |
| Impacts: We are able to use a cryptographic library that is both strong and fast. |

# Methodology

## Programming languages

Our password generation algorithm is implemented in Javascript. Our manager is implemented in Python using Tkinter, a framework that allows us to deploy our product straight onto the user's desktop. The reason we use it is because it is simple to design and easy to deploy, a good library to deploy a desktop app within a short frame of time.

Our database system is custom-programmed, which has the unique feature of being integrated into the end product instead of being stored in a server, eliminating the need for network communication and thus securing us against data interception. Moreover, it is also secured in AES-256 with a unique seed that changes periodically.

In analyzing our passwords, we have opted to use the SciPy toolkit, due to its support for data-oriented applications such as web-crawling, statistical analysis, and custom APIs such as zxcvbn and Selenium, which we will expound on in the Analysis section. Its high readability also makes changes and additions easier for a highly modular data extraction and analysis program.

## Algorithm

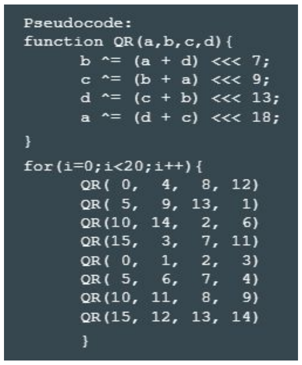
All random generation is handled through the Stanford Javascript Crypto Library, providing cryptographically secure random number generation through the use of hardware entropy collection, a technique to generate random data through receiving input from the environment. This greatly increases the security of our passwords since they become less predictable.

Our algorithm can be explained as such in short:

1) Accept a random seed from the user. Generate a random seed otherwise.
    a) This is to increase the inherent randomness of the algorithm in order to generate a better unique password.

2) Process the string with the Salsa20 algorithm

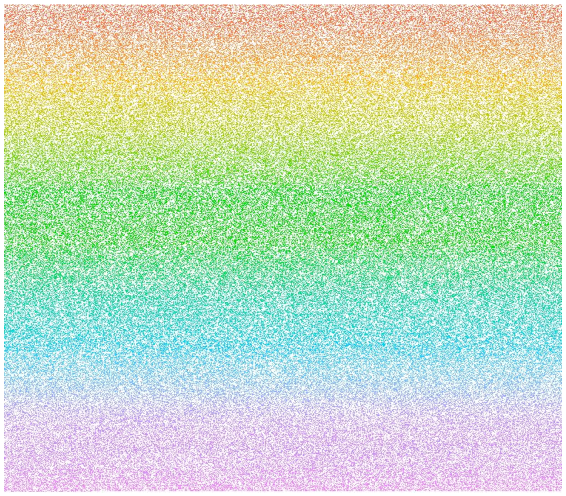
a) This encryption algorithm provides the bulk of our security, taking our seed and performing bitwise operations for every set-length section of the seed such as XOR operations, bitwise shifting and rotations. This program has been demonstrated to be extremely secure for commercial cryptographic applications, with weaker variants (Salsa20/8 and Salsa20/12) being reported to have at least a $2^{249}$ - time complexity attack, being equivalent to needing an eternity (2.866 * 10^55 years, assuming 1 trillion attacks per second). Hence it is the perfect encryption algorithm to process our seed.



```
Pseudocode:
function QR(a,b,c,d){
        b ^= (a + d) <<< 7;
        c ^= (b + a) <<< 9;
        d ^= (c + b) <<< 13;
        a ^= (d + c) <<< 18;
}
for(i=0;i<20;i++){
        QR( 0,  4,  8, 12)
        QR( 5,  9, 13,  1)
        QR(10, 14,  2,  6)
        QR(15,  3,  7, 11)
        QR( 0,  1,  2,  3)
        QR( 5,  6,  7,  4)
        QR(10, 11,  8,  9)
        QR(15, 12, 13, 14)
        }
```

Pictured: Pseudo-code of the Salsa20 algorithm

3) Hash with the MurmurHash3-128 algorithm
    a) This 128-bit hashing algorithm provides us with a fast method to format our processed seed into a 32-character string. It is unusually suited to creating such unique strings due to high randomness and a low collision rate.



Pictured: Randomness distribution diagram of the MurmurHash3 algorithm. One can easily observe its evenness.

4) Format the hash according to user requirements

a) Lastly, we make alterations to the string such as truncating characters and expanding our character set in order to customize the password to the user's requirements.

## Product Design

### Password Generator

Our first final product is a password generator which can help users create unique passwords, with them being able to choose the character set, whether to include symbols and the password length. Fig. 3.1 shows the screenshot of the user interface of the generator.

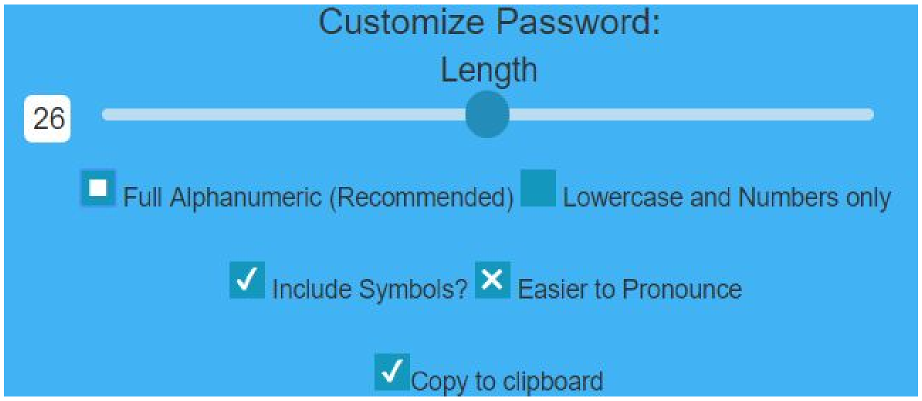


Fig 3.1: Our password generator

### Password Manager

Our second final product is a desktop app implemented in Python, allowing users to manage their passwords safely without exposing their data to the internet.

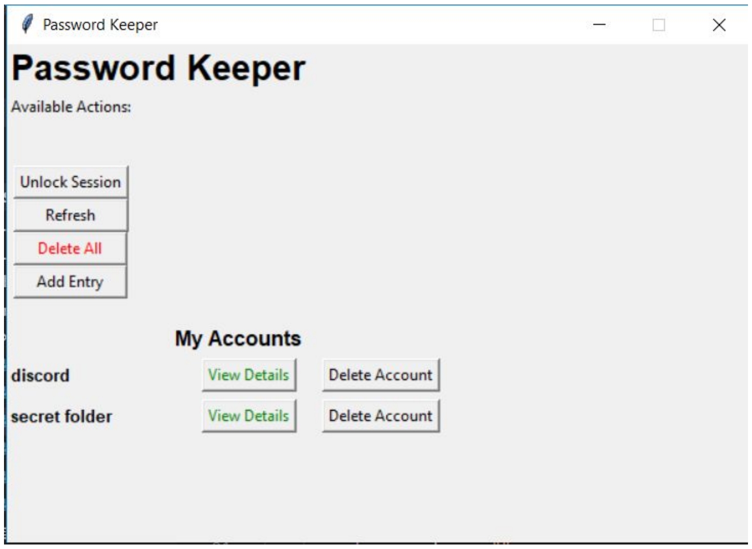


Fig 3.2: Password Generator.

## Features

Our app and password generator include several features to guard against password retrieval techniques:

1) Our app operates entirely offline, removing opportunities for the password to be retrieved from the Internet.
2) We offer displayless transfer of passwords so that passwords can never be seen by shoulder-surfers.

We initially intended to integrate these products. However, the password generator lost much of its versatility (could only be used within the password generator if it was to be made more convenient)  and security was greatly compromised(see Findings under Analysis).

Our products are also optimised for the end user by being easy to navigate and having a simple display.

# Analysis

## Methodology

We conducted a comparative analysis between the password generated by us and those of commercial password generators using various algorithms.

For this application, we have opted to use zxcvbn, a library developed by Dropbox that actually simulates dictionary-style and brute-force attacks, checking all password combinations starting with the most common ones and their variations, for example anagrams and 13375p34k( a form of Internet slang where letters are replaced with numbers and words are spelt to sound as they are spoken), succeeding in rooting out weak and reused passwords where other checkers fail.

We collected passwords using libraries as Selenium, WebDriver, and pandas. Selenium allows us to crawl data from websites that are generated in part by Javascript. Since most password generators are implemented in Javascript, this is the perfect library for our uses. WebDriver allows us to interact and automate the browsing of websites for our use of Selenium, such as re-generating passwords and configuring password settings like the inclusion of symbols and upper-case letters. pandas allows us to format the data we collect into an Excel file format.

We then compared passwords from LastPass, Dashlane, and Avast by noting the time required to conduct an analysis through zxcvbn. The library provided the number of guesses to $\log_{10}$, so we had initially planned to use this metric. However, after a trial run we found that the number of guesses required was too high to be recorded properly.

| | CybAware | LastPass | Dashlane | Avast | CybAware (Python) |
|---|---|---|---|---|---|
| Avg Time/ms | 4800 | 2750 | 3267 | 3169 | 2974 |
| Max Time/ms | 15741 | 8775 | 16068 | 9772 | 13825 |
| Min Time/ms | 1963 | 1216 | 1752 | 1633 | 997 |
| Above Average / # of Passwords | 435 | 414 | 438 | 381 | 402 |

Table: Data collected on 26-character passwords, sample size = 1000.
All passwords are guaranteed to be alphanumeric with symbols, case-sensitive.

## Findings

1) Passwords generated by us required 47% more time on average to be attacked than the nearest competitor.
2) Our passwords required 12% more time to be attacked in the worst case than the nearest competitor.
3) Our passwords were more consistent overall, coming in second at 435 passwords being above average.
4) We have attempted a version of our algorithm but in Python, in order to make the password generation process more convenient. However, the security cost was too great to make it worthwhile.

# Future developments

Our password generator and manager still have some areas of improvement:

1) Our password length is capped at 32 characters as a direct consequence of our algorithm. This limits our capability to provide stronger passwords, although not crucial to providing greater security as alternatives forsake the innate strength of the algorithm for length, actually reducing the possibilities for attackers to attempt in a brute-force situation.
2) Our generator and manager are not integrated for versatility and technical reasons, but an iteration of a password manager featuring our generation algorithm would make the manager more convenient to use.

## Project Timeline

| Date | Agenda | Action Performed |
|---|---|---|
| March | Project group founded | - Founded project group, idea proposed to our mentor<br>- Research conducted into cryptographic algorithms |
| April | Change of idea | - Conception of a password generator with additional safety features and a custom algorithm |
| May-July | Implementation & Testing | - Implemented password generator algorithm in HTML, Javascript, and CSS<br>- Comparative analysis conducted on generated passwords against commercial generators like LastPass and Dashlane |
| July-Aug | Improvements | - Taking into account the judges' suggestions, we implemented a complementary password manager in Python. |

# Conclusion

Overall, our results show that our password generating algorithm is effective and consistent in generating secure passwords which are harder to crack by various methods such as brute-force, frequency and dictionary attacks as compared to other commercial password generators. Thus, our project is an improvement from previous algorithms and will help users better protect themselves against password-related attacks.

# Bibliography

1. An Update On Taking Steps To Protect Our Members. (n.d.). Retrieved from https://blog.linkedin.com/2012/06/09/an-update-on-taking-steps-to-protect-our-members
2. Baharudin, H. (2019, March 21). Passwords and usernames of staff from MOH, MOE and other agencies stolen and put up for sale by hackers. Retrieved from https://www.straitstimes.com/singapore/compromised-log-ins-passwords-from-several-govt-agencies-on-sale-online-says-russian-cyber
3. Password Reuse. (n.d.). Retrieved from https://xkcd.com/792/
4. Dwolfhub. (2019, May 29). Dwolfhub/zxcvbn-python. Retrieved from https://github.com/dwolfhub/zxcvbn-python #scoring.py
5. Bernstein, D. J. (2007). *The Salsa20 family of stream ciphers* (Unpublished master's thesis). The University of Illinois at Chicago. Retrieved April 20, 2019, from https://cr.yp.to/snuffle/salsafamily-20071225.pdf.
6. Cimi. (n.d.). Cimi/murmurhash3js-revisited. Retrieved from https://github.com/cimi/murmurhash3js-revisited
7. VowpalWabbit. (n.d.). VowpalWabbit/vowpal_wabbit. Retrieved from https://github.com/VowpalWabbit/vowpal_wabbit/wiki/murmur2-vs-murmur3
8. Ian Boyd (n.d.). Which hashing algorithm is best for uniqueness and speed? Retrieved from https://softwareengineering.stackexchange.com/questions/49550/which-hashing-algorithm-is-best-for-uniqueness-and-speed
9. Weir, M. (1970, January 01). New Paper on Password Security Metrics. Retrieved from https://reusablesec.blogspot.com/2010/10/new-paper-on-password-security-metrics.html
10. Grassi, P. A. et al(n.d.). Retrieved from https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-63b.pdf
11. De Carne de Carnavalet, X., & Mannan, M. (n.d.). Retrieved from http://users.encs.concordia.ca/~mmannan/publications/password-meters-tissec.pdf
12. Ghorbani Lyastani, S. et al(n.d.). Retrieved from https://arxiv.org/pdf/1712.08940.pdf
13. Salsa20. (2019, April 25). Retrieved from https://en.wikipedia.org/wiki/Salsa20
14. Issue 552749 - chromium - An open-source project to help move the web forward. - Monorail. (n.d.). Retrieved from https://bugs.chromium.org/p/chromium/issues/detail?id=552749
15. Leonhard, M. D. (2006, October 30). A Comparison of Three Random Password Generators. Retrieved May 16, 2019, from https://pdfs.semanticscholar.org/e6da/b609de895e1752e96691da9023ed1994b27e.pdf