Memorable Password Generation with AES in ECB Mode

Timothy Hoang[1] and Pablo Rivas

[1]Marist College, Poughkeepsie New York 12601, USA, timothy.hoang1@marist.edu

Abstract This paper presents ALP: an adjustable Lexicon-generated password mechanism. ALP is a password encryption web application. In ALP, a password is generated randomly from an imported dictionary of words. For example, with an entire dictionary, it will pick out random strings that add up to a 128-bit hex key and return the key to the user. The password will be "easy to remember" since it will be made of a chosen pool of words. It will then take in the user's message to be encrypted and output the encrypted message according to the key. The password generator, if need be, can be switched to a completely random mode which will make it output a string of random numbers and letters if the user does not choose to use the easy to remember password generator function. The application will also be able to decrypt the AES-128 encrypted message and transform it back into normal text. This experiment suggests that the proposed model is a block cipher that can successfully generate a memorable random password.

Keywords: AES, ECB, Password Generation

1 Introduction

Password security is a constant bother in the modern world. Today, it is possible to save passwords into Google's autofill and other password manager programs. These programs can generate passwords on the fly and store them on a computer or online account so that one would only have to remember a single password to access every other password they have. This is useful for people to have a multitude of different secure passwords for all of their accounts to make it more difficult for malicious entities to access them all. However, the downside of this technology is that all of the accounts are linked to one crucial location. Another downside is needing to first log into the master account if signing onto one of the accounts from another location, making access to the desired account difficult. The last downside is that many of these programs charge a subscription fee to securely store passwords. With ALP program, it is possible to create multiple randomly generated passwords that should be easy to remember given the user's parameters. This solves the one location and access from other computers issues since one can more easily remember each individual password since they are readable. Since the password will be readable, and therefore more memorable, one will have an easier time accessing their accounts from other locations without access to the password manager.

2 Concerns

There is a concern for dictionary attacks with normal random word combinations for password generation [1]. That is why the lexicon was made completely customizable, as it will negate the effect of this type of brute force attack. With the customizable lexicon, the user is not limited to traditional words found in the dictionary. The user is able to add whatever string that they deem memorable. This includes slang, names, made-up words, non-English words, sentimental numbers, and any bit of information. This is useful as it allows one to create passwords that are equally complex as a completely random string of the same length, while also being secure from traditional dictionary attacks since a stylistic touch is added to the lexicon. Therefore, unless the attackers know exactly what "words" were in the dictionary at the time of creating the password, they cannot perform a dictionary attack. This makes the customizable lexicon approach protected against brute force dictionary attacks.

E.g., in the lexicon, the user puts in these strings, "4Plakilt3, WaR75atel8, M1zule, Laz4Apt, M0der0ck". To the user, these made up words are memorable for some reason (the numbers, capitalization, and segments of the words hold some significance to that specific user and no one else). Now, say the user requests that the password be 16 characters in length. From those words, the program can combine them to generate a password such as M1zuleWaR75atel8 or 4Plakilt3Laz4Apt. So long as the user has enough words and variety in word lengths within the dictionary, it is possible to create many completely unique passwords at any length. Since the "words" in the dictionary are deemed memorable according to the user, what may look like a bunch of random characters to someone, can be easily remembered by the user. Unless the attacker knows every word within the lexicon that the user utilized at the time of generating their password, the only way to brute force it would be to brute force every single character. This gives the same time complexity as if they were to brute force a password where every character is random.

3 Methodology/Experimental Setup

The lexicon reading and password generation portion of the ALP application is complete. The function is for the program to read a lexicon that the user creates. From the lexicon of words, a random “readable” password will be generated. Each word in the .txt file is separated by line. Although any length word can be added to the lexicon, the program, as it stands, will only choose words that are 16 characters or less since that is what is required for the AES 128. This character count can be expanded easily in the code, but for the purpose of demonstrating its use in an AES 128 cipher, it was limited to 16. If the smallest word in the lexicon is too big to fill in the remaining characters needed, randomly generated characters will be chosen for the remainder of the password. In addition to this, the user is able to toggle between a “read- able” and a completely random password. In the completely random passwords, each character is randomly generated and not read from the lexicon at all. These programs are only intended for English letters and numbers, so it may produce problems when introduced to other characters.

For the decryption portion, inverse programs were created for the ShiftRow(), NibbleSub(), and MixColumn() functions that are present in the AESencryption.java file which encrypts messages with AES 128. The changes to these functions include the change present in InvMixColumn(), where there is Galois multiplication in different fields. In addition to these inverse functions, the order of key operations has been reversed.

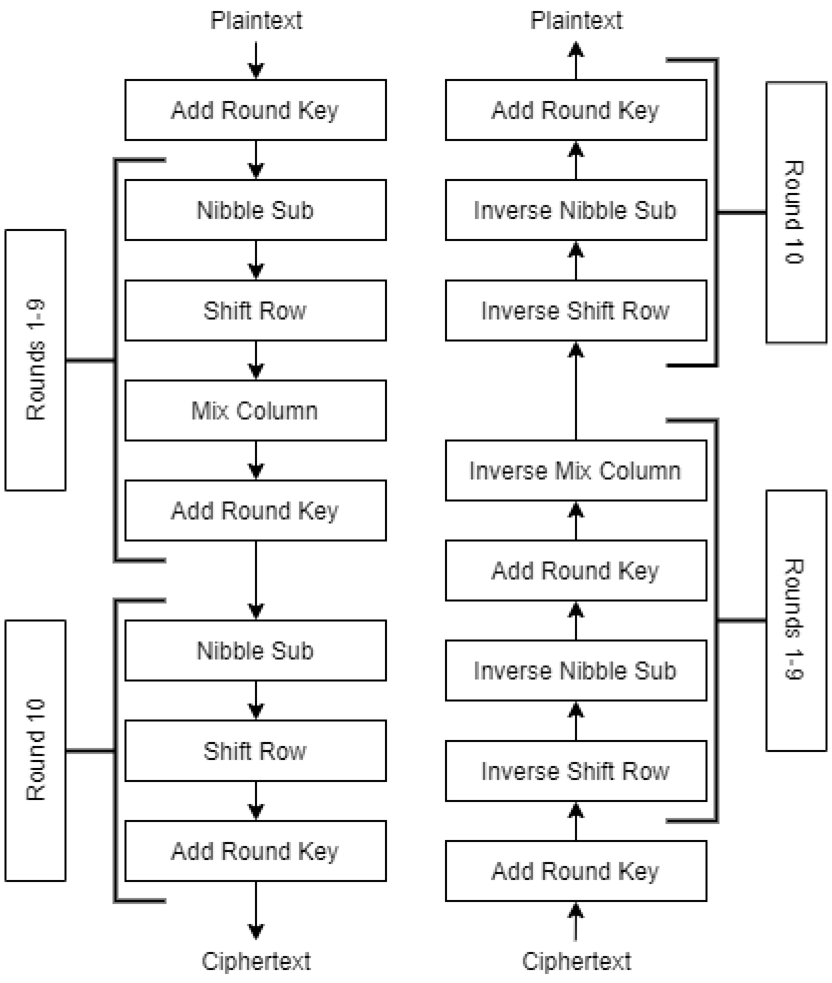


Fig. 1. Order of operations for AES encryption and decryption.

Fig. 1 contains the order of operations to be done for encryption and decryption in AES. Since the encryption part was completed previously, here is an explanation of the right side of the picture from the bottom up. The program starts by adding the round key. Then for the next nine keys the following will take place: invShiftRow(), invNibbleSub(), add the round key, then InvMixColumn(). For the last key, the program will do the same process except it will leave out the InvMixColumn() function. The order of operations is achieved in the AESdecrypt() function, which is located in the AESdecipher.java file [2].

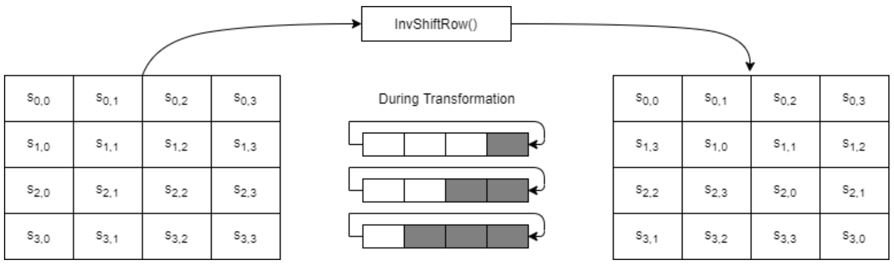
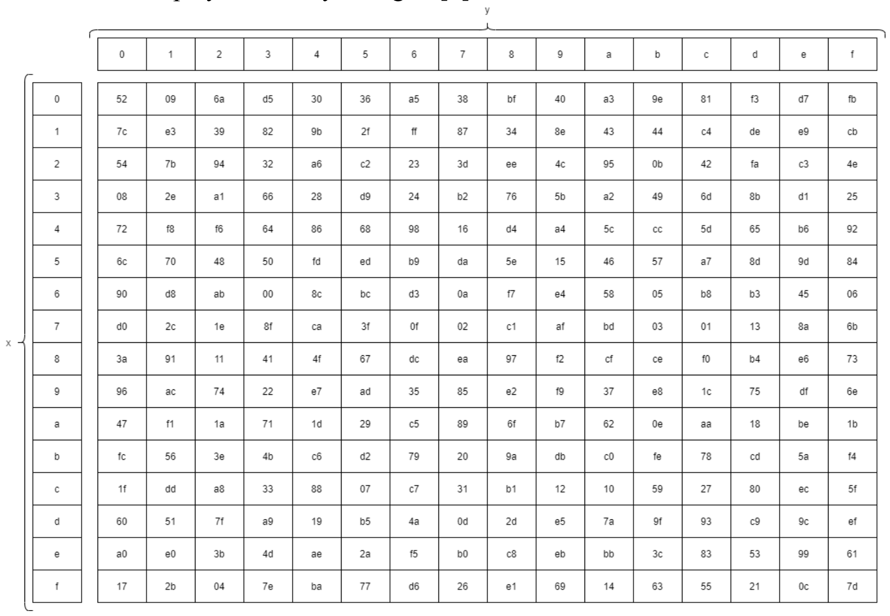


Fig. 2. Inverse Shift Row function.

In InvShiftRow(), all the program does is change the rows opposite to how it was shifted in ShiftRow(). That is, instead of taking the last one, two, and three subjects of the bottom three rows in the matrix and bringing them to the front, the program takes the first three, two, and one subjects from those rows and moves them to the back. This is displayed visually in Fig. 2 [3].



	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Fig. 3. Inverse Nibble Substitution function substitution values.

For InvNibbleSub(), the program does the exact same operation as AESNibble-Sub() but uses the substitution values contained in the table shown in Fig. 3 [4].

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \times \begin{bmatrix} \text{galois(14)} & \text{galois(11)} & \text{galois(13)} & \text{galois(9)} \\ \text{galois(9)} & \text{galois(14)} & \text{galois(11)} & \text{galois(13)} \\ \text{galois(13)} & \text{galois(9)} & \text{galois(14)} & \text{galois(11)} \\ \text{galois(11)} & \text{galois(13)} & \text{galois(9)} & \text{galois(14)} \end{bmatrix} = \begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \end{bmatrix}$$

Fig. 4. Inverse Mix Column function.

For InvMixColumn(), the program will perform operations as shown in Fig. 4, where the numbers in the second matrix are the Galois field numbers [5].

4 Experiment Results

The program will produce a memorable password using words provided in the lexicon of choice. By running the main program, it has produced passwords such as hic3cco7dinhl4mb and contr2buto9hedrd. These passwords are concatenations of the strings ["hic", "3cco7dinh", "l4mb"] and ["contr2buto9", "hedrd"] respectively. The test cases for each of the inverse functions can be performed by running test.java program. test.java will display the matrices passed into each of the functions as well as their resulting matrices. The output for these test cases can be found in the testcases.txt file located within the data folder of this project. All of the inverse functions perform as intended. Attempting to run the full decryption process does not decrypt the message properly. Despite the fact that several experiments were conducted to address this, the source of this issue has not been found.

5 Conclusion

In conclusion, the ALP program is able to create secure, memorable passwords for use with the lexicon of choice. However, the decryption process of the program is still being investigated. Work has been made on the individual processes and order methodology functioning properly as far as the test cases and results show, there is something holding back the full decryption. The password generation portion of ALP has many use cases for individuals of all kinds, as everyone needs proper security for their many accounts in the modern world. For further improvement, figuring out and fixing the error concerning the order of operations in the AES decryption process is the most crucial. In addition to this, adding the ability to have special characters in the lexicon for password generation and being able to set requirements for the resulting password, such as requiring a certain number of numbers, special characters, and capitalization can be done.

The code to reproduce the experiments can be accessed under the MIT license in this repository: github.com/timhoangt/ALP

Acknowledgements

This work was supported in part by the New York State Cloud Computing and Analytics Center, and by the VPAA's office at Marist College.

References

1. Bošnjak, Leon & Sres, J. & Brumen, B.. (2018). Brute-force and dictionary attack on hashed real-world passwords. 1161-1166. 10.23919/MIPRO.2018.8400211.
2. Wadday, Ahmed & Wadi, Salim & Mohammed, Hayder & Abdullah, Ali. (2018). Study of WiMAX Based Communication Channel Effects on the Ciphered Image Using MAES Algorithm. International Journal of Applied Engineering Research. 13.
3. Bhattarai, Bibek & Giri, Naresh Kumar. (2015). FPGA Prototyping of the secured biometric based Identification system. 10.13140/RG.2.1.4067.2729.
4. Selimis, Georgios & Kakarountas, Athanasios & Fournaris, Apostolos & Milidonis, Athanasios & Koufopavlou, Odysseas. (2007). A Low Power Design for Sbox Cryptographic Primitive of Advanced Encryption Standard for Mobile End-Users. Journal of Low Power Electronics. 3. 327-336. 10.1166/ jolpe.2007.139.
5. Raju, L.M. & Manickam, Sumathi. (2015). Secured high throughput of 128-Bit AES algorithm based on interleaving technique. 10. 11047-11058