Alex Fuoco
MSCS 630 Security Algorithms
Project -- Note Encrypter

Introduction

Modern Applications rely on data from users, as the rate of data accumulation increases there are increasing risks to the security of the data. The robust types of data from users including the user's location to their food preferences are available with the types of data collected. From web browsers like Google Chrome, to Samsung Smart TVs the collection of user data has increased and is not ending in the near future. Internet of Things (IoT) devices do not give the user the ability to choose when and how their data is being collected. (Kuchler, 1) The collection of user data and availability of the data has led to a system that has encroached user privacy. As applications collect sensitive data, the storage of the data in an encrypted way is primary for security. So recently as 2012, Facebook stored user passwords in plaintext, allowing for thousands of users passwords to be accessed by employees without security access. (Kastrenakes, 1) Which is just one of the many security problems Facebook has faced in the previous years. Solutions to these problems include the user having access to secured and encrypted applications to store their data. Users deserve a space to control their data, and choose the way their data is stored.

Background

When encrypting data there have been many types of algorithms made. There are symmetric encryption algorithms, which allow for decryption. Also there are hashing algorithms for encryption, which is a one way function that cannot be decrypted. For our use case, we want to decrypt our note so a hashing algorithm would not be the correct choice, hashing algorithms are a better choice for storing passwords. There are downsides to hashing algorithms, as CPUs get more advanced the algorithms are easier to break, which is the reason why hashing algorithms such as MD5 are not suitable for use. The MD5 hash has rainbow tables, which is a table of all the possible inputs and their corresponding outputs. Another popular algorithm is RSA or public key encryption, which could be an adequate choice for this note encrypter. Although when comparing the results of AES vs RSA, the amount of resources and time used for RSA on texts is more than double the time needed for encrypting with AES. (Del Pozo, et al. 537) AES has also been combined with other encryption algorithms to enhance the security. (Aljawarneh, et al. 22708)

AES is a symmetric key-block cipher that uses a block length of 128 bits. (Rachh, et al. 1766) This algorithm uses a 16 byte key. The proper technique for AES 128 bit is to go through 10 rounds with multiple steps each. The initial round adds the round key to the state, The next 9 rounds first substitutes the bytes, normally with an S-Box lookup table, then shifts the rows, and mixes the column with Galois Field arithmetic, and adds the round key. The 10th and final round substitutes the bytes, shifts the rows and adds the final round key and outputs the cipher text. (Aljawarneh, et al. 22711) For keys, the input is a 16 character hexadecimal string that is converted to a 16 byte array for encryption.

Implementation

The application proposed will include AES 128-bit for encryption. The use case for this application includes personal notes that will be encrypted in storage and displayed on the web application. This problem of not storing the data encrypted is seen in many modern applications, not even 10 years ago Facebook was still storing passwords as plaintext, not hashed at all. The ever increasing security risks to our data should be solved in a way that prioritizes the user. This application is a storage web app allowing the user to input a 16 character hexadecimal string, their note and a title to encrypt. This is then stored and decrypted only with the Hex Key provided. This web application will allow the user to store notes in an encrypted way and knowing only those with the provided Hex Key will have access to their note.

When thinking of the application the use cases can be sharing messages with others, or storing text documents in an encrypted way. This web application provides a fast and efficient way of encrypting and decrypting data. This key hex is never stored, meaning once encrypted the person with the key will only be able to access it. This provides protection to the users notes, if the database gets accessed there will be no way to decrypt the stored notes. Having the user only able to access their notes gives security in the back end. The title field is the only field not encrypted on storage. This will allow the user to know the note they are accessing, so titles should never give personal information out.

The web application is a ReactJS web application, using MaterialUI components styling. The storage of the notes is through DynamoDB on AWS, the table only includes a UUID as the index, the encrypted note, and the plaintext title. This allows for a minimal amount of user information stored as it is not attached to a user personally. The library used for AES encryption was aes-js, this library allows for implementing AES at 128-bit, 192-bit and 256-bit. We chose AES 128-bit allowing the user to input a 16 character hexadecimal key, which was then converted to a byte array before encrypting. As previously mentioned the key string was not stored. To use 192-bit or 256-bit AES encryption the hex key length would have to be changed to 24 and 32, respectively. Being an easy change for future implementations.

When decrypting the note, the user will input the key in hex. If the key is the correct length and the correct type, then the note will attempt to be decrypted. If the key is incorrect the note will still be decrypted, although the output will be incorrect. If the key is of the wrong type and length, the user will be alerted.

Testing

For test cases, the main cases involve the encrypting and decrypting of the notes. This will involve both retrieval and input into DynamoDB. For the encryption the test cases include long and short notes, the title and keystring are also needed. The key string test cases will include the correct hexadecimal key string with a length of 16, and the incorrect key string inputs of wrong length and wrong type. The decryption will also include inputting the correct key string and incorrect key string, including the wrong length and type, or wrong key string in general. Below are the example test cases and the expected output.

Test Type	Title	HexKeyString	Note	Encrypted Note	Output
Input	Test 1 - Short	1234567890abcdef	This is a test with a short length	4271535ffc37e1aebd5e972914f6bfb4dc8e7f2e26f763002f547b09596985aa6c23	Inserted the note
Expected Return	Test 1 - Short	1234567890abcdef	This is a test with a short length	4271535ffc37e1aebd5e972914f6bfb4dc8e7f2e26f763002f547b09596985aa6c23	This is a test with a short length
Wrong Key	Test 1 - Short	1234567890abcdee	This is a test with a short length	4271535ffc37e1aebd5e972914f6bfb4dc8e7f2e26f763002f547b09596985aa6c24	獉z画 +KF‸句 b□H愜劣
Wrong Length Key	Test 1 - Short	12345678900	This is a test with a short length	4271535ffc37e1aebd5e972914f6bfb4dc8e7f2e26f763002f547b09596985aa6c25	WRONG KEY INPUT

Wrong Type Key	Test 1 - Short	1234567890abcdRZ	This is a test with a short length	4271535ffc37e1aebd5e972914f6bfb4dc8e7f2e26f763002f547b09596985aa6c26	ERROR DECRYPTING THE MESSAGE

Conclusion

The Note Encrypter application is a way for users to take control of the storage and collection of their data in a way that prioritizes their safety of data. Implementing the symmetric key block cipher AES 128-bit for encryption and decryption allows for proper security for the users notes. Having the user input the hexadecimal key string without storage of the key string allows the user the sole way to decrypt their input, or those who the user gave access to the key string. Implementing AES at 128-bit allows for the least amount of complexity, but gives the opportunity to enhance in the future. The Note Encrypter is a solution to the problems of users not having control of the way their data is stored and gives the user the ability to control the way their data is used.

References

Aljawarneh, Shadi, et al. “A Resource-Efficient Encryption Algorithm for Multimedia Big Data.” Multimedia Tools and Applications, vol. 76, no. 21, Springer US, 2017, pp. 22703–24, doi:10.1007/s11042-016-4333-y.

Del Pozo, Ivan, and Mauricio Iturralde. “CI: A New Encryption Mechanism for Instant Messaging in Mobile Devices.” Procedia Computer Science, vol. 63, Elsevier B.V, 2015, pp. 533–38, doi:10.1016/j.procs.2015.08.381.

Kastrenakes, Jacob. “Facebook Stored Hundreds of Millions of Passwords in Plain Text.” *The* Verge, The Verge, 21 Mar. 2019.

Kuchler, Hannah. “The Internet of Things: Home Is Where the Hackers Are.” FT.com, The Financial Times Limited, 2017.

Rachh, Rashmi Ramesh, et al. “Efficient Implementations for AES Encryption and Decryption.” *Circuits, Systems, and Signal Processing*, vol. 31, no. 5, SP Birkhäuser Verlag Boston, 2012, pp. 1765–85, doi:10.1007/s00034-012-9395-0.