Credit Card Transaction Implementing Format Preserving Encryption(FPE)

Group 5: Sai Aka , Konduru Sandilya , Jasti Pardhav

December 3, 2021

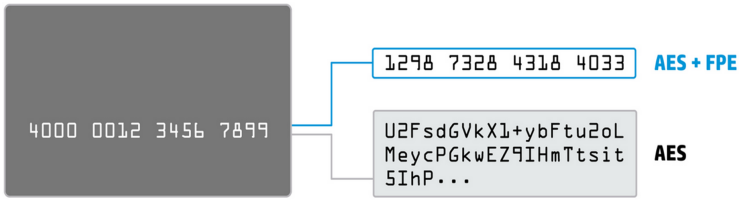
Abstract

Format Preserving Encryption (FPE) schemes encrypt a plaintext into a ciphertext while preserving its format (e.g., a valid credit card number is encrypted into a valid credit card number), thus allowing encrypted data to be stored and used in the same manner as unencrypted data. Motivated by the always increasing use of cloud-computing and memory delegation, which require preserving both plaintext format and privacy. we implemented a simple credit card processing system using FF3-1 format preserving encryption scheme based on recent 2019 NIST standards in python.

1 Introduction

Now a days, as more and more applications take shape to facilitate online shopping, money transfer and small scale enterprises are moving to third party services to store their data and associated applications, there arises a need to secure online transactions, and a need for protect data from anomaly administrators and malicious attackers.

Encrypting Credit card numbers (CCN), Social Security Numbers (SSN) in huge legacy databases has become a very complex task if it has the same problem to change existing database schema. There are many challenges First, the cost of modifying existing databases. Second, sensitive information like SSN and CCN are used as a primary key in database changes in this field may require significant schema changes. Third, applications are also related to specific data format, will require a format change. Format preserving encryption (FPE) is a solution to the above problems.



To meet these demands, Traditional as well as new and improved cryptographic methods have been constructed to achieve greater security. But, a major problem in adopting these methods is the requirement to change the existing databases to incorporate the encrypted data. This problem can be handled with a Format Preserving Encryption. With FPE, encrypted data will gain its original format.

The remaining sections of the paper is organised as follows: section deals with related work on FPE where we will discuss about existing methods for constructing FPE. Section 3 deals with technical details of FPE. Section 4 deals with summary of the project and section 5 is conclusion.

2 Background

Format Preserving Encryption (FPE) is a deterministic encryption with a randomized key generation(keyGen: K$\rightarrow$ k). The format of the plaintext(N) is to be given beforehand. So that the ciphertext will be of the same format. One of the downsides of this is that the same encryption scheme is not useful for different formats of plaintext.

Formally, Encryption E: k $\times$ t $\times$ N $\times$ M $\rightarrow$ $M \cup \perp$

Where t, M are tweak and Message spaces, respectively.

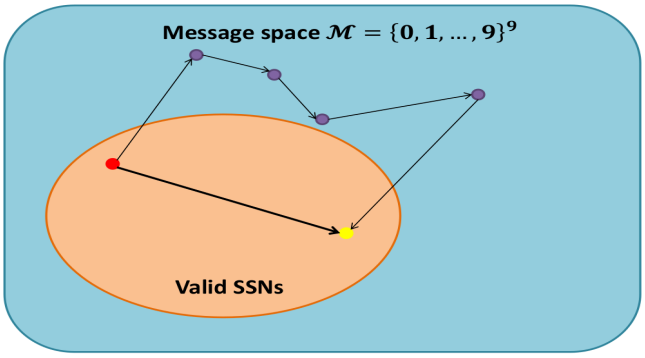
There are many ways of constructing FPE. Some of them are as follows:

2.1 Prefix Cipher[1][2]:

To create an FPE algorithm using prefix cipher on (0,...,N-1) is to assign a pseudorandom weight to each integer, then sort by weight. The weights are defined by applying an existing block cipher to each integer.

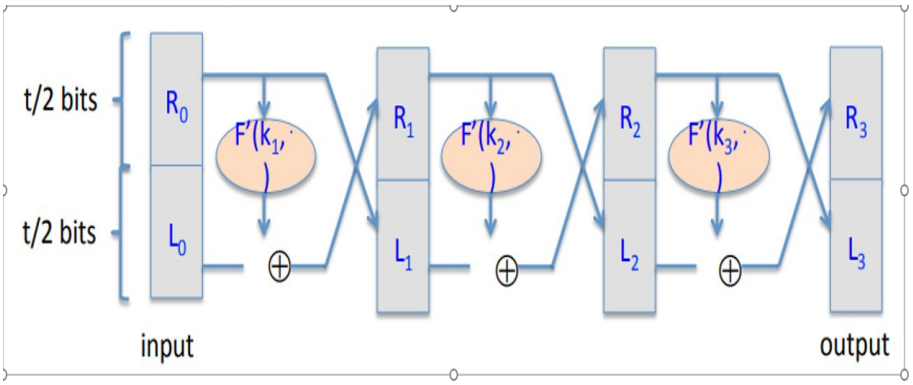
2.2 Cycle Walking[1][2]:

If we have a set M of allowed values within the domain of a pseudorandom permutation P (for example P can be a block cipher like AES), we can create an FPE algorithm from the block cipher by repeatedly applying the block cipher until the result is one of the allowed values (within M).



2.3 Feistel Mechanism[1][2][3]:

One way to implement an FPE algorithm using AES and a Feistel network is to use as many bits of AES output as are needed to equal the length of the left or right halves of the Feistel network.If a 24-bit value is needed as a sub-key, for example, it is possible to use the lowest 24 bits of the output of AES for this value.This may not result in the output of the feistel network preserving the format but iterating the feistel network can result in maintaining the format.



2.4 FF3-1[4]

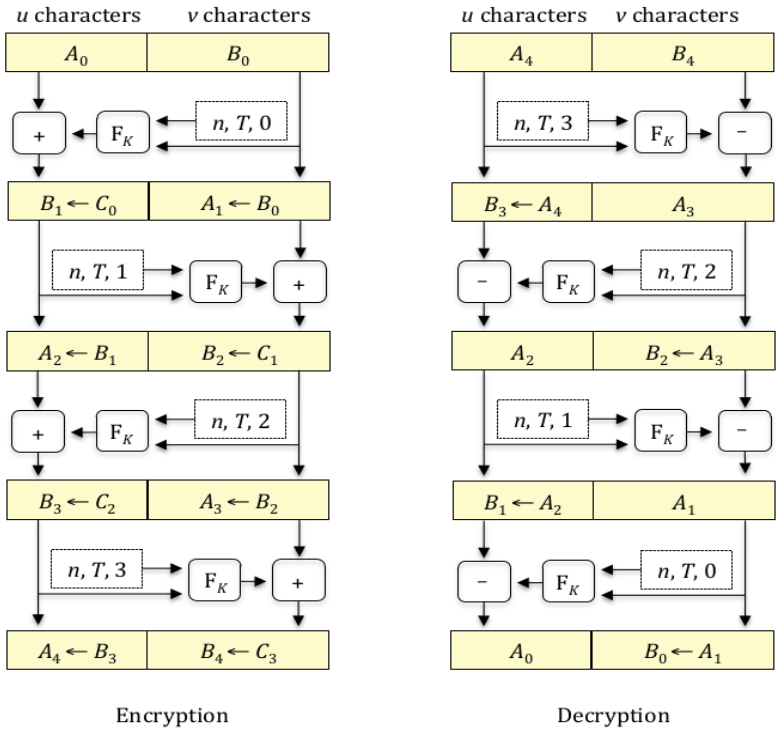
FF3-1 scheme is based on the Feistel structure. The Feistel structure consists of several iterations, called rounds, of a reversible transformation. The transformation consists of three steps:

1) The data is split into two parts.

2) A keyed function, called the round function, is applied to one part of the data in order to modify the other part of the data.

3) The roles of the two parts are swapped for the next round.

FF3-1 consists of 8 rounds.



3 Technical Details

We built our code in python. We maintain two python files in which one file acts as client(can be considered as POS machine) and another one as server. We used

socket programming for communication between the client and server. Every successful transaction is reflected in a separate text file named as database.txt.

Apart from client, server and database we have another ff3-1 python file which is implemented based on NIST 2019 standard [3]. We used AES encryption in ECB mode to implement FF3-1 standard. Although ECB mode is semantically insecure usage of Feistel network shadows the insecurity of ECB mode.

Our approach was to encrypt the card using a predefined tweak shared between both the bank and POS terminal where the secret key is generated by using the pin as the seed for the pesudorandom genator provided by python core modules.

One improvement we would like to provide is that the tweak should be dynamic and set by bank server for every transaction instead of being a fixed one for protection against CCA attacks and in general we would advise to either replace the legacy system all together as it would be economically cheaper to trade for more secure and advanced systems than withstanding a breach or use new encryption method of format preserving tokenization.[4]

In brief format preserving tokenization is where a token is created using third party service providers or server itself. The property of these tokens is that the tokens have no meaningful value when they are breached and they are also time sensitive, when tokenization is implemented in combination of FPE the security of system goes up on par of modern software although it's computationally expensive especially on server side

Tokenization schemes can use either single-use or multi-use tokens. This is similar to the distinction between randomized and deterministic encryption. With single-use tokens, every time a the same input data is used, a new token is generated. With multi-use tokens, using the same input data results in the same token[4][5].

The primary difference and benefit of using tokenization is that tokenized data cannot be returned to its original form. Unlike encryption, tokenization does not use keys to alter the original data. Instead, it removes the data from an organization's internal systems entirely and exchanges it for a randomly generated nonsensitive placeholder (a token). These placeholders can be stored within an organization's internal systems for business use while sensitive values are safely stored outside of its environment.

So, in the event that a tokenized environment is breached, no sensitive data or compromising keys/credentials would be revealed—only the nonsensitive tokens. Because no sensitive data is being stored, none is available to be stolen. In effect, the risk of data theft is virtually eliminated.

4 Summary

Health care industries, Credit card industries require data to be stored in specific formats in their database. The format of the data needs to be preserved even after encryption of the data. Using general encryption schemes makes the encrypted data format invalid for their operation. The database schema of the data again needs to be changed. The solution for this problem is Format Preserving Encryption.

We used current NIST approved FPE standard i.e, FF3-1 encryption algorithm to implement a simple credit card transaction system in python which consits of a server and client communicating via socket. Every successful transaction gets updated in a separate database text file.

FPE is not a right choice for industries dealing with sensitive data as the encryption is a deterministic encryption. The alternate method for this Format Preserving Tokenization. Unlike encryption, tokenization cannot be decrypted. It is just a random string of same format as data. One downside, is that tokenization requires more storage than compared to FPE. Therefore, usage of Format Preserving Encryption or Format Preserving Tokenization depends upon the which type of data is handled. i.e, more Sensitive or less sensitive.

5 Conclusion and Future Work

We have learned the importance of using strong encryption like AES to create a custom encryption which suits the needs of systems which are outdated but of vital importance and we were also able to survey the newly developed cryptography techniques in finical sector, we wish that if we instead of working on our own method of FPE instead of starting outright with NIST standard, this would definitely save a lot of time and give us more time to work on end user experience and UI

For Future work as there are no public implementation of ff3-1 algorithm the work we have done could be made public as a open source project and be beneficial for others, this needs the code to be cleaned a bit and made production ready, at the same time the credit card systems UI implementation would be fun to work on and would definitely give a sense of gratification.

6 References

1. 6th BIU winter School, http://cyber.biu.ac.il/event/the-6th-biu-winter-school/

2. Format-Preserving Encryption. Mihir Bellare, Thomas Ristenpart, Phillip Rogaway and Till Stegers, 2009.

3. NIST.SP.800-38Gr1 2019.

4. Differences between Format Preserving Encryption and Format Preserving Tokenization, https://fortanix.com/blog/2018/01/secure-legacy-applications-using-format-preserving-encryption-tokenization/

5. The Difference Between Format-Preserving Encryption and Tokenization. https://www.comforte.com/fileadmin/Collateral/comforte_FS_tokenization_vs_FPE_WEB.pdf