A Comparison of HMAC-based and AES-based FFX mode of Operation for Format-Preserving Encryption

Levent Ertaul, Jalaj Neelesh Shah, Sofiane Ammar
California State University, East Bay, Hayward, CA, USA
levent.ertaul@csueastbay.edu, jshah22@horizon.csueastbay.edu, sammar@horizon.csueastbay.edu

***Abstract* — As usage and importance of smart phones and tablets grow, apps have come to dominate digital media. With limited computation capacity of mobile devices, performance plays a vital role in providing good user experience to the apps. This in conjunction with the recent security breaches leading to millions of stolen credit cards, makes it essential to ensure confidentiality while maintaining high performance. This paper presents performance comparison of AES (CBC) and HMAC (SHA-1) based PRFs for FFX mode of Format Preserving Encryption for a mobile app that functions as a credit card wallet.**

I. Introduction

Recent security breaches into various US retailers like Target [1], Home Depot and 7-11[2] not only indicate financial losses but also highlight the vulnerability of financial-information systems.

According to The Nilson Report 2013,[3] Credit Card Frauds around the world have grown from $2.5 billion to $7.5 billion in the last decade (as we can see in figure 1.1). While the growing trend continues this decade too, it has sharpened. Between 2010 and 2012 alone, there was a growth of $3.5 billion. It is only expected to grow in the coming years. This makes secure storage of Credit Cards or all cards for that matter even more important.

Figure 1.1 Global Card Fraud.[3]

Credit cards are stored in encrypted form. The encryption technique used i.e. Format-Preserving Encryption (FPE) [14] is slightly different than the regular encryption techniques. FPE encrypts plaintext of a particular length and format into ciphertext of the exact same length and format. For instance, encrypting a 16-digit Credit Card Number (CCN) using FPE would give a 16-digit number. FPE is a rapidly emerging cryptographic tool in applications like financial- information security in legacy databases. It becomes vital for structured data such as CCNs and Social Security numbers as the databases expect them to be in the exact same format and of exact same length for data-level encryption. As shown in figure 1.2, a regular encryption scheme like AES [15] would result in ciphertext of characters and varied length, FPE would give us ciphertext that would seem to look like a genuine CCN.

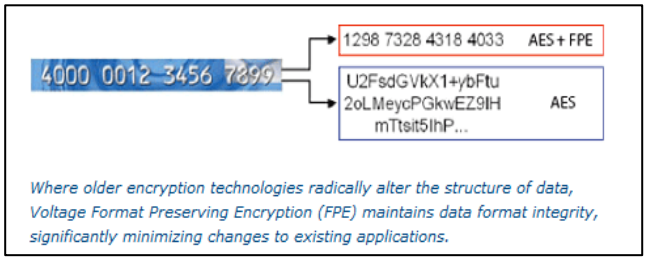

Figure 1.2 Format Preserving Encryption against Regular AES [4].

This also serves as a means to disguise intruders as it becomes difficult to distinguish between real CCNs and encrypted CCNs.

Wherever money is involved, security must be high and rightly so. Confidentiality is the most important thing while dealing with credit cards. In encryption mechanisms, it is generally true that increasing number of rounds increases quantitative security. While storing Credit Cards, we would ideally want as many numbers of rounds as possible, but, increasing the number of rounds would make the encryption process slower. Earlier, we said that performance is very important from a user point of view. Thus, a right balance has to be attained so that none is compromised.

Figure 1.3 shows that Mobile devices have out taken Desktops in terms of numbers globally. Smart phones and Tablets account for 60% of time spent on digital media in the US. The same report also suggests that it is 'usage of apps' that leads to this trend as 52% of this time is spent on apps alone. This also marks the decline of web dominance.

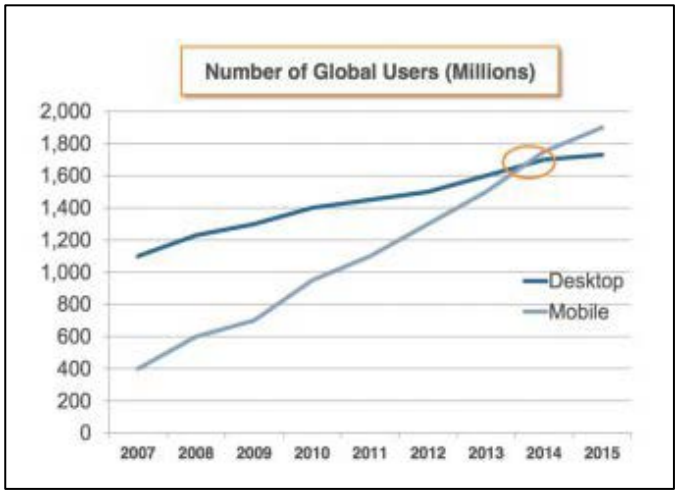

Figure 1.3 Number of Global Users of Digital Media.[5]

Smart phones and Tablets still do not match up with the computation capacity of desktops. For users to hang on to apps, good user experience is essential which can be provided if the app is fast enough. There are various apps like 'Google Wallet'[16] for storing Credit Cards and making transactions.

A lot of resources are put into such apps and as momentum shifts towards such e-payment systems, we reckon it is essential to have a good balance between security and performance. While there are no known weaknesses of FPE [13] (if parameters chosen correctly), not much work has been done to test performance of FPE. This paper compares various round functions for FFX mode[6] of FPE and presents quantitative results. These results would help in choosing the right round function so that the overall algorithm is fast.

In the section II, we look at the basic algorithm in which various parameters are given and the Encryption process is explained. Section III gives specifications of the implementation followed by the tests and results in section IV. Finally we lay out the conclusion.

II. ALGORITHM

A. *Mode of Operation*

We chose the FFX mode for FPE given by Bellare, Rogaway and Spies [6] as it is an extension to FFSEM [18] and supports tweaks that prevent dictionary attacks. FFX is defined as Format Preserving Feistel-based Encryption. The 'X' stands for parameter profile, which in our case is A10.

B. *Tweak*

Tweak's literal meaning is to alter or to modify. Tweak is defined as a set of unrelated mappings by the authors of FFX.[6] The idea behind tweaking is that while Issuer Identification Number (IIN)[12] for different Credit Card issuers ensures that the first few digits for each of them are different, the remaining digits can still be identical. This could lead to Dictionary attacks [17]. Thus, it is recommended to tweak some of the middle digits with the remaining ones.

In our algorithm, we tweak the middle eight digits with the starting four and the last four. We, however, do not use unrelated mappings to tweak. We use arithmetic and logical operations over the middle eight digits with the combination of first and last four digits. In this way, we believe we are bringing in more variation. For instance, for a Visa [22] card starting with 4, the original paper would only have one tweaked output per mapping, while with our tweak it could be anything from 0 to 9 depending on the fifth digit.

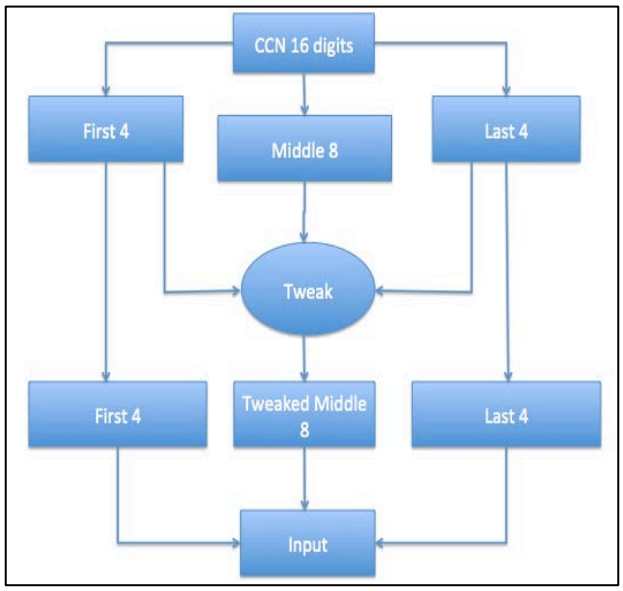

Figure 2.1 Working of Tweak

As we can see in figure 2.1, the tweaked middle eight digits are then combined together with the original first four and the last four. This together goes through the FFX.Encrypt (shown in figure 2.2).

C. *Round Function*

The Round Function that is basically a Pseudo Random Function (PRF) can be constructed from a Block cipher or a Hash Function. AES and HMAC [19] are recommended for Block cipher and hash function respectively [6]. We use CBC mode[20] for AES-based round function while SHA-1[21] for HMAC-based round function.

D. *Parameter Choices*

We use Parameter collection A10 as our implementation is based on 16 digit decimal numbers. Another set of parameters known as Parameter collection A2 is to be used for binary inputs.[6] Parameter Choices for A10 are given in Table I.

TABLE I. PARAMETER COLLECTION A10

Parameter	Choice
Radix	10
Key	128, 192, 256-bit keys
Addition	Blockwise
Method	1
Split	8
Rounds	12

E. *FFX.Encrypt*

The tweakedCCN, Tweak and the Key are then passed on to the main encryption function (figure 2.2). The tweaked CCN is split into two halves. The right half is hashed with SHA-1 based HMAC using a secret key. The hashed right half is then added (blockwise) to the left half. This becomes the right half for the next round while the right half of the last round becomes the left half of the next. The process goes on for twelve rounds until the two halves are finally merged.

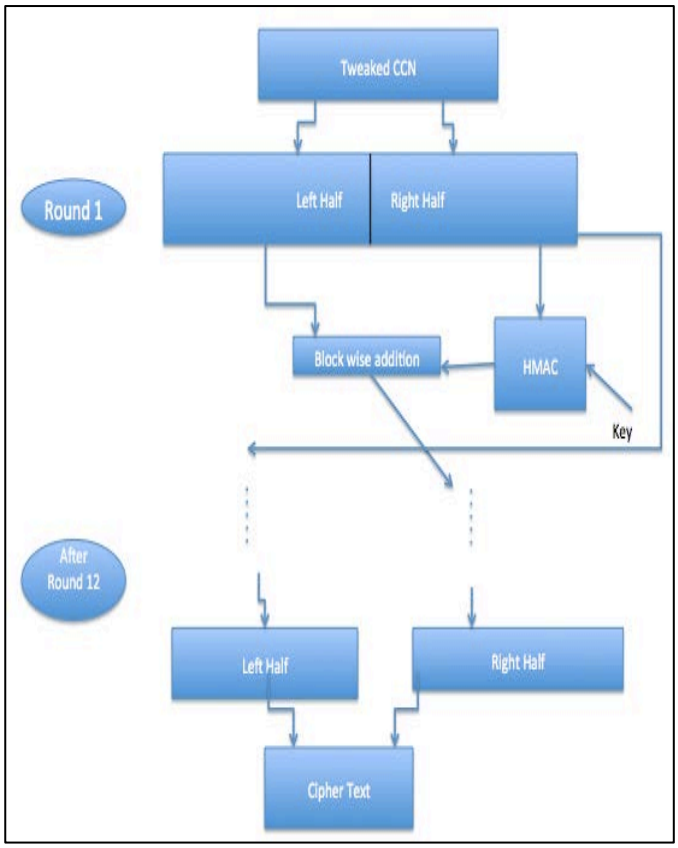

Figure 2.2 One complete cycle of unbalanced Feistel-based FFX.Encrypt

F. *Cycle Walking*

Cycle Walking is essential to FFX as it ensures that the ciphertext from FFX is of the desired format. With respect to our algorithm, this essentially means that if we assume that we want to encrypt an American Express [23] card. We know that the IIN for American Express is either '34' or '37'. In order to maintain its format, the ciphertext should also start with '34' or '37'. FFX alone cannot guarantee this. It has to be used in conjunction with Cycle Walking or Dense Encoding [6]. We choose Cycle Walking. As soon as FFX.Encrypt terminates, it is checked if the ciphertext falls within the set of VALIDCCN(X) that specifies the validity predicate. (For which American Express would be 34XXXX and 37XXXX). If yes, the algorithm terminates, otherwise FFX.Encrypt is called upon the result of the first cycle.

III. IMPLEMENTATION

A. *Specification*

The detailed Hardware and Software specifications are given in Table II and Table III respectively.

TABLE II. HARDWARE SPECIFICATION

Type	Specification
Type of System	64-bit Operating System
Processor	Intel® i5 Quad-Core 2.5GHz
Memory	4GB RAM

TABLE III. SOFTWARE SPECIFICATION

Type	Specification
Operating System	Windows 8
IDE	NetBeans 8.01[27], Android Studio[26]
Programming language	Java
Runtime Environment	JRE 6
Development Kit	JDK 1.7.0.45
Database	MySQL 6.1
Network Model	TCP/IP Client/Server Model
Crypto Library	Java.Security

B. *Screenshots*

We implemented a mobile wallet that can store encrypted credit cards. The screenshots are taken on Android Studio.

On launching the app, it would ask for a four-digit access pin (figure 3.1). It is done to avoid unauthorized access to the wallet. This four-digit pin can be set up at the time of installing the app.

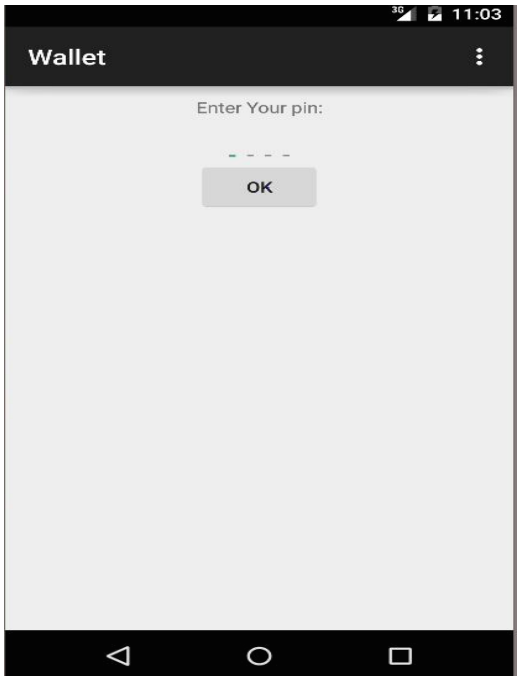

Figure 3.1 Access Page

If the pin is verified, the user is logged in (figure 3.2). The user can now see current credit cards that the wallet holds or the user can add a new card.

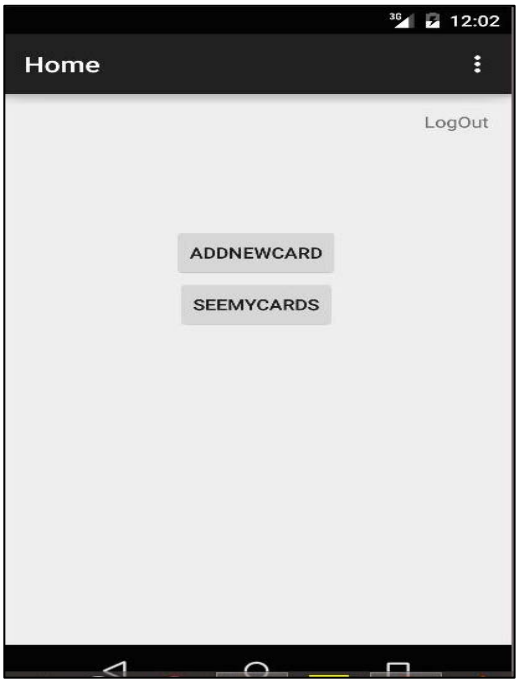

Figure 3.2 Home Page

Let's assume that the user taps on 'ADDNEWCARD.' The next screen will take the card details in the manner shown in figure 3.3.

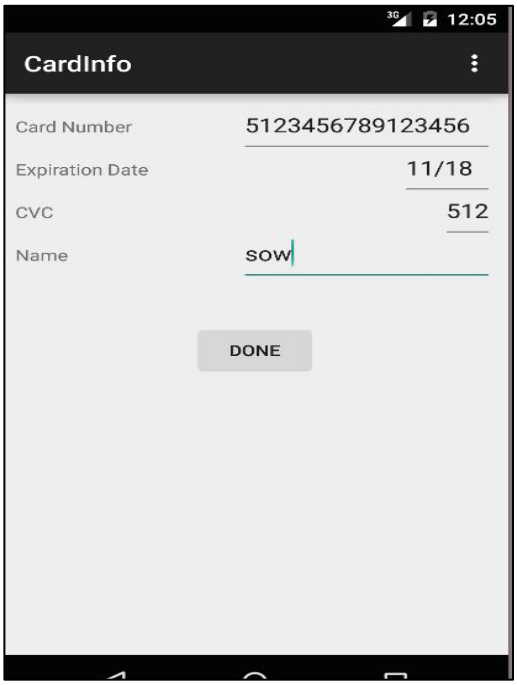

Figure 3.3 Add Card Details Page

For user convenience and ease in remembering, the user can nick name the newly entered card (figure 3.4).

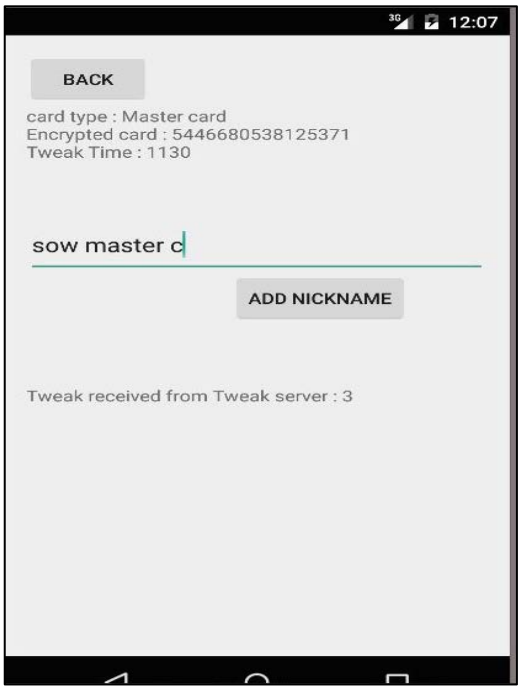

Figure 3.4 Nick Name page

The user is now taken back to the Home page from where current cards can be seen by tapping 'SEEMYCARDS' as shown in figure 3.5.

Figure 3.5 Stored Card Display Page

IV. TESTS AND RESULTS

For testing purpose, simulation of a sample size of 1000 or 4000 on android based mobile phone was not possible due to limited memory on mobile devices. The file containing log of CCNs could not be processed. Thus, we used CPU clock to time the performance of various combinations on the system specified in the above tables. We timed FPE only so as to get a precise measure of the performance of the algorithm itself by removing anomalies due to lag in Client-Server model and Database connections.

While we ran tests for big samples on Windows system having much more computation capabilities, we also ran test for very small sample sizes on Android Studio as well. Simulation on mobile emulator, showed no notable deviation from the performance seen on computer system. This could be due to the fact that the mobile device configured on the emulator did not have any other resources taken by the system.

A. *Comparison of AES CBC v. HMAC (SHA-1)*

The basic motivation of the paper was to find out that among the two round function candidates i.e. CBC mode of AES and HMAC, which one performs better. We used 256-bit key on a sample of 4000 credit cards. HMAC SHA-1 shows 67% better performance than AES CBC (Figure 4.1)

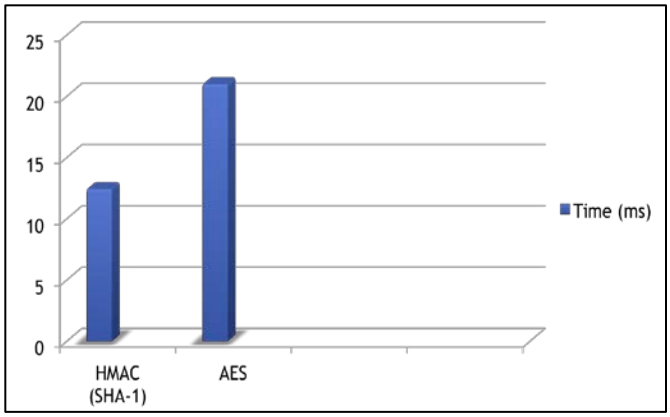

Figure 4.1 AES CBC v. HMAC (SHA-1)

B. *Comparison between different key sizes: 128-bit v. 192-bit v. 256-bit*

We can see in figure 4.2 that there is little difference in performance of SHA-1 when different key sizes are used. It is because of the change in number of cycles that each run took. On the first look of it, it gives an idea that changing key size affects the number of cycles. After several runs, we can conclude that variation in key size does not affect performance and that the number of cycles was completely random and independent of key size.

Performance figures according to benchmarks[9] suggest that as we increase key size for AES CBC, the performance deteriorates. However, in our tests the results (figure 4.3) were surprising. 192-bit key size showed dramatically good results. The random number of cycles again played a role in this.

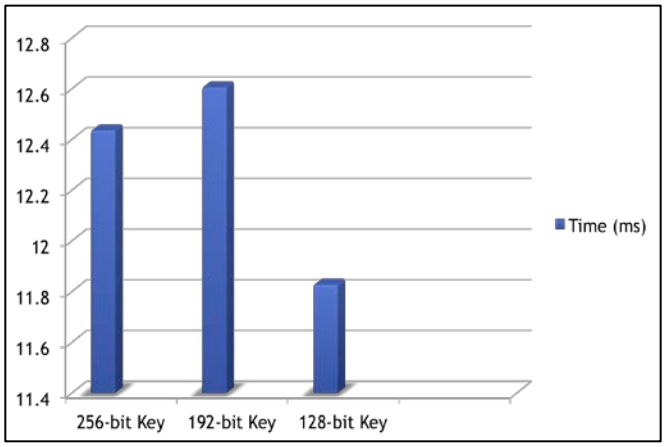

Figure 4.2 128-bit v. 192-bit v. 256-bit keys

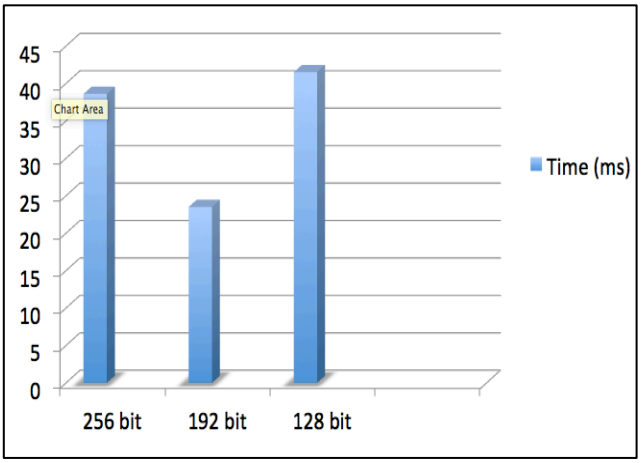

Figure 4.3 128-bit v. 192-bit v. 256-bit keys

C. *Comparison based on Credit Card Issuers: Visa v. MasterCard[24] v. Discover[25] v. American Express*

In order to look at the practical aspect of the implementation, we ran tests on different samples each limited to credit cards issued by a particular company. Typically one would assume that, the greater the fixed number of digits for a credit card, the higher the constraint on Cycle Walking, thus, the algorithm would go through more number of cycles. As a result, the number of fixed digits at the beginning of a CCN that vary as the issuer varies, alters the time that the encryption would take. We ran the tests through both round functions i.e. AES CBC and HMAC SHA-1.

As we can see in figure 4.4 and 4.5, the results are no different than expected. Visa shows the best performance as its Issuer Identification Number (IIN) is 4.[13] Just one condition has to be satisfied, thus, fewer cycles. With MasterCard the IIN is 51-55. With American Express, the IIN is '34' and '37.'[10] Thus, American Express takes more time and cycles as despite the equal number of conditions on MasterCard and American Express, the latter has a stricter choice between two digits only. While the IIN for Discover is '6011' [11], the four conditions take a toll on the performance of the algorithm that it practically crashed most times. Thus, we relaxed the Cycle Waking constraint to '60'.

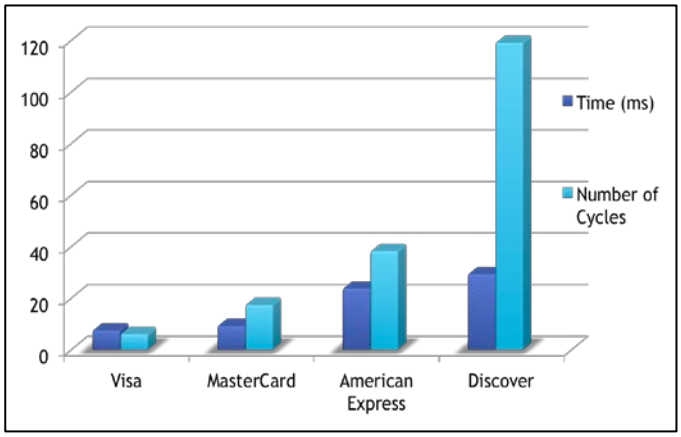

Figure 4.4 Visa v. MasterCard v. Discover v. American Express

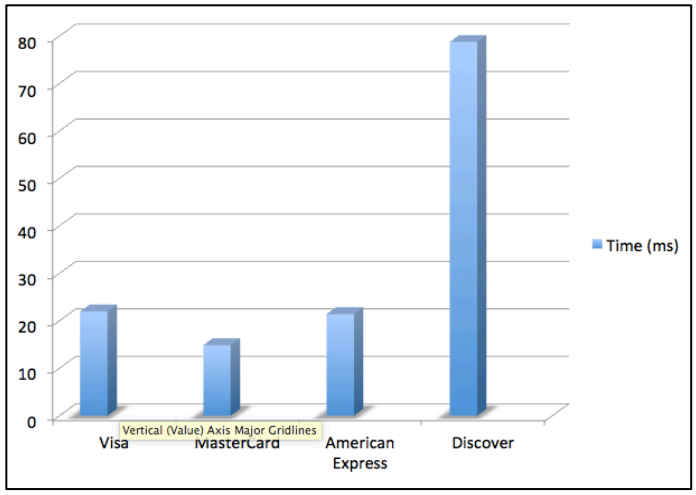

Figure 4.5 Visa v. MasterCard v. Discover v. American Express

D. *Comparison of SHA-1 v. SHA-256[21] v. SHA-512[21]*

It is proven that it takes a complexity of less than 80 to find collisions in SHA-1[7]. If Moore's law [8] holds still until mid-2020s, the computation power would be 2 times from what it is now. Thus, there is ample evidence why we need to migrate from SHA-1 to SHA-2 [21]. In alignment with this, we extended our tests to SHA-256 and SHA-512. As we can see in figure 4.6, SHA-512 performs better. According to the performance benchmarks [9], one would expect SHA-1 to be the fastest. However, SHA-1 based HMAC as a round function takes more number of cycles. While the difference between the number of cycles taken by SHA-256 and SHA-512 is not much, SHA-512 is much faster [9]. Although SHA-256 is slower (per round) than the rest, it takes lesser number of rounds and thus shows better performance than SHA-1.

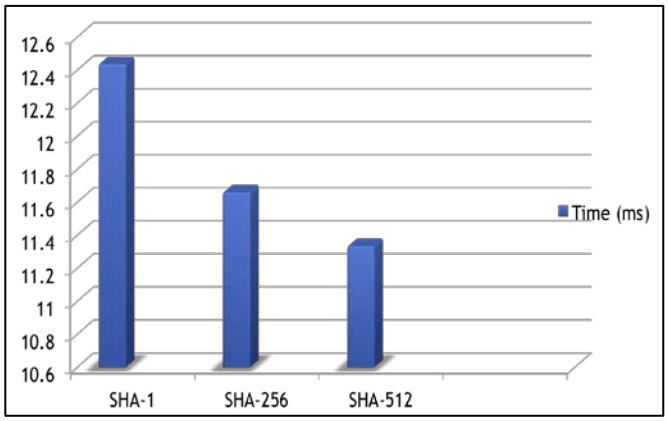

Figure 4.6 SHA-1 v. SHA-256 v. SHA-512

V. CONCLUSION

In this paper, we compared how different round functions for FFX line up in terms of performance. We also tested for different key sizes as migration from 128-bit keys to 256- bit keys has already initiated.

We also show how using different credit card companies

affect performance. Discover cards took a toll on performance as it took more cycles to give the ciphertext because of the longer IIN. Visa took the least number of cycles and thus least time.

While we use SHA-1 for HMAC-based round function, we also extend our implementation for SHA-256 and SHA-512.

We come to the conclusion that HMAC is a good candidate for FFX in terms of performance. It outruns AES by almost 67%. We recommend using SHA-512 for implementing HMAC as it shows promising performance and has fewer collisions as well.

We hope that more enhanced mobile wallets are launched in the future and the results presented in this paper help the designers.

VI. ACKNOWLEDGEMENT

We gratefully acknowledge the assistance and participation of Jil Trivedi, Sowjanya Kosaraju: Math and Computer Science, CSU East Bay. Jil and Sowjanya contributed towards implementation and testing of AES (CBC) based design.

VII. REFERENCES

[1] Sara Germano, Robin Sidel, Danny Yadron. “Target Faces Backlash After 20-Day Security Breach.” The Wall Street Journal. 19 Dec 2013. Web. 19 Mar 2015.

[2] Dan Goodin. “TJX suspect indicted in Heartland, Hannaford breaches.” The Register. 17 Aug 2009. Web. 18 March 2015.

[3] “Global Card Fraud.” 2013 Nilson Report. Aug 2013. 18 March 2015.

[4] “Preserving Critical Business Functions by Maintaining Data Format” Voltage security.

[5] Adam Lella, Andrew Lipsman. “The U.S. Mobile App Report” comScore. 21 Aug 2014. Web. 18 March 2015.

[6] Mihir Bellare, Phillip Rogaway, Terence Spies. “The FFX Mode of Operation for Format-Preserving Encryption.” NIST submission. 20 Feb 2010.

[7] Xiao Yun Wang, Yiqun Lisa Yin, Hongbo Yu. “Finding Collisions in Full SHA-1.” NSFC Grant No. 90304009.

[8] Moore, Gordon E. "Cramming more components onto integrated circuits" (PDF). Electronics Magazine. 1965.

[9] "Crypto++ 5.6.0 Benchmarks". 1 April 2009. Web. 18 March 2015

[10] "Card Security Features" (PDF). American Express. January 2001.

[11] "Discover Network - IIN Range Update, 8.2" (PDF). September 2008.

[12] “Identification cards -- Identification of issuers.” Part 1: Numbering system. ISO/IEC 7812-1:2006.

[13] Phillip Rogaway. “A Synopsis of Format-Preserving Encryption”. 27 March 2010. University of California, Davis, CA, USA.

[14] John Black and Philip Rogaway, “Ciphers with Arbitrary Domains”. Proceedings RSA-CT, 2002, pp. 114–130.

[15] Joan Daemen, Vincent Rijmen, "The Design of Rijndael: AES – The Advanced Encryption Standard". Springer, 2002. ISBN 3-540-42580-2.

[16] "Coming soon: make your phone your wallet". Official Google Blog. May 26, 2011. Retrieved April 22, 2015.

[17] R. Shirey. “Internet Security Glossary”. May 2000. RFC 2828.

[18] Terence Spies. “Feistel Finite Set Encryption Mode”. NIST.

[19] H. Krawczyk, M. Bellare, R. Canetti. “HMAC: Keyed-Hashing for Message Authentication”. February 1997. RFC 2104.

[20] NIST Computer Security Division's (CSD) Security Technology Group (STG) (2013). "Block cipher modes". Retrieved April 22, 2015.

[21] NIST. “Secure Hash Standard (SHS)”. FIPS PUB 180-4.

[22] “Visa Inc.”. visa.com. Retrieved April 22, 2015.

[23] “American Express”. Americanexpress.com. Retrieved April 22, 2015.

[24] “MasterCard”. Mastercard.com. Retrieved April 22. 2015.

[25] “Discover Financial”. discover.com. Retrieved April 22, 2015.

[26] “Android Studio”. developer.android.com. Retrieved April 22, 2015.

[27] “Net Beans”. netbeans.org. Retrieved April 22, 2015.