Web Tap Payment Authentication and Encryption With Zero Customer Effort

Henry Ng

Tap-Card-Pay Systems Corporation, Vancouver BC V5X3Y3, Canada
henryng@tapcardpay.com

Abstract. We propose a public-key authentication and encryption application that secures the messages between Tap-Card-Pay application, Tap-Card-Pay Systems Corporation, customers, and merchants allowing the customer to complete transactions without requiring the customer to input sensitive information. With authentication and encryption, the application transfers the credit card information from the smartphone's near field communication device onto the merchant webpage. Security weaknesses are also presented to show how to attack this design.

Keywords. Public-Key Cryptography, Digital Signatures, Applications.

1 Introduction

In our Tap-Card-Pay application (app), we remove the keyboarding from the customer and add the simplicity of tapping the credit card on the smartphone when the customer tries to checkout on the merchant's webpage. However, with this much simplicity, there is a cost on the computing resources needed, rather than human effort and intervention, to securely handle sensitive information from the customer's credit card.

The purpose of our security application is to avoid exposing the credit card information to adversaries, after the customer's name, numbers, and expiry date is read from the smartphone operating system application programming interface (API), which originally reads from the near field communication (NFC) of smartphone [2].

The Tap-Card-Pay app sends the message, containing the encrypted credit card information, to the merchant webpage when the customer executes the app to read and send credit card information. In order for Tap-Card-Pay app to trust the merchant, Tap-Card-Pay Systems Corporation needs to certify the merchant, producing a signed certificate, and the Tap-Card-Pay app needs to authenticate this certificate.

Digital signatures are messages that use some secret known to the signer and are based on the content of the message being signed [1]. The application verfifies the trustworthiness of the public-keys using digital signatures. Certification of a merchant's public-key by Tap-Card-Pay Systems enables authentication, data-integrity, and non-repudiation. During certification, Tap-Card-Pay Systems would need to digitally sign the merchant's public-key.

In public-key encryption, an entity has a public-key and a private-key, where the public-key can be known by others and the private-key needs to be secret [1]. Public-keys can be used to encrypt data and also to verify digital signatures [1].

For each transaction, a new, unique keypair is generated and a copy of the merchant's public-key is downloaded onto the Tap-Card-Pay app.

1.1 Paper Organization

In the next sections, we provide details on the public-key signature and encryption application. Section 2 lists the design requirements. Section 3 shows the key exchange methods between entities. Section 4 shows how the app handles the cryptography and how to verify the app. Section 5 shows the flow order during a transaction. Finally, in section 6, we describe attacks on the application.

2 Requirements

During a transaction, the customer is required to input the credit card information on the merchant webpage. Normally, this input is on the form by typing in the credit card information.

With Tap-Card-Pay, the customer taps the credit card on the NFC to input the credit card information. In an ideal situation, the NFC data containing the credit card information is entered instantly on the webpage without transporting plaintext through an untrusted environment where there can be any application or system processes running on the smartphone that can capture plaintext. Refer to Figure 1.

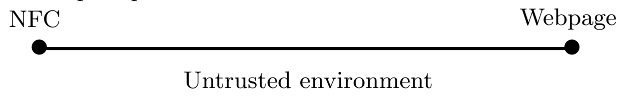
We use authentication and encryption in our application to securely transport messages from the NFC to the webpage. Merchant web servers host the webpages.

2.1 Limitations

We list the limitations that restrict what the proposed security application can do.

1. The customer cannot remember passwords or secrets.
2. The app cannot store plaintext passwords or secrets.

Fig. 1. The untrusted environment is between the NFC read and the merchant webpage. We need to transport the credit card information between these two locations. It is unsafe to transport plaintext credit card information.



3. The merchant cannot store or verify credit card information before the customer submits the credit card information on the merchant webpage.
4. The customer should not type any credit card information into the merchant webpage.

3 Key Exchange

We define a way to exchange public-keys between the merchants and Tap-Card-Pay Systems.

3.1 Merchant

The merchant generates a keypair $Priv_M$ Pub_M and gets the public-key Pub_M signed by Tap-Card-Pay Systems creating digital signature Sig_T.

For each transaction, the merchants generates a new, unique keypair with private-key $Priv_J$ and public-key Pub_J. The Pub_J, which will be introduced to the app, is signed by the merchant's private-key $Priv_M$. Since Pub_M is trusted, once it is signed by Tap-Card-Pay Systems, Pub_M can introduce new public-keys.

3.2 Tap-Card-Pay Systems

Tap-Card-Pay Systems generates a keypair $Priv_T$ Pub_T for signing and authenticating the public-keys from the merchants. Pub_T is installed in the Tap-Card-Pay app.

3.3 Customer

The customer does not have any keypair. The private-key would need a password protection.

4 App Methods

The app uses two cryptographic methods, authentication and encryption. Authentication is a service related to identification and two entities entering into a communication should identify each other [1]. We use encryption to ensure privacy, keeping information secret from all but those who are authorized to see it [1]. In addition, we suggest that the customer compares the hash of the app before use. In the following subsections, we describe the details.

4.1 Authentication

During authentication of the merchant's public-key Pub_M, the Tap-Card-Pay app verifies the digital signature Sig_T that is attached to Pub_M using verification Pub_T. If the signature is incorrect, the app rejects the public-key Pub_M and notifies the customer. Otherwise, Pub_M is considered to be probably valid from the merchant.[1]

Once Pub_M is considered probably valid, then Pub_M is used to authenticate Pub_J, which is signed by the private-key $Priv_M$. Signature Sig_M that is attached to Pub_J is verified. Pub_M is the trusted introducer of Pub_J and it introduces Pub_J to the app for later encryption operations.

4.2 Encryption

In our application, the receiving merchant has public-keys Pub_M, Pub_J and private-keys $Priv_M, Priv_J$. The purpose of $Pub_M, Priv_M$ is to introduce public-keys Pub_J to the app. Our application encrypts the credit card information using public-key Pub_J, public-key encryption, symmetric key K, and block cipher.

K is pseudorandomly selected for keying a block cipher which maps n-bit plaintext blocks to n-bit ciphertext blocks. Blocks of credit card information is encrypted using the block cipher to produce ciphertext c_1.

In addition, the symmetric key K is encrypted using Pub_J and public-key encryption to produce ciphertext c_2.

The application sends the message containing c_1, c_2 to the webpage for decryption. Using $Priv_J$, public-key algorithm, and block cipher, c_2 is decrypted to produce K and c_1 is decrypted to produce credit card information, in sequence.

4.3 Ciphertext Message Passing

After the customer's name, numbers, and expiry date is read from the smartphone operating system API, which originally reads from the NFC of smartphone hardware, encryption is applied on the information producing ciphertext. The ciphertext is sent from the app to the *localhost* MySQL[2] database, which is running as a system service in Android. Next, the ciphertext is retrieved by a *localhost* PHP[3] script, which is hosted on *localhost* lighttpd[4], through SQL statetment information requests to the MySQL database. There is no decryption method before the ciphertext reaches the webpage.

The PHP script presents JSON[5] data to the JavaScript[6], which is hosted on the merchant website. The JavaScript decrypts c_1, c_2 using $Priv_J$, public-key algorithm, and block cipher.

[1] We show attacks in a later section.
[2] http://www.mysql.com/
[3] http://www.php.net/
[4] http://www.lighttpd.net/
[5] http://www.json.org/
[6] http://www.w3schools.com/js/

Fig. 2. Transportation of public-keys and digital signatures. The merchant sends the $Pub_J, Sig_M, Pub_M, Sig_T$ to webpage and app for verification and encryption. c_1, c_2 are sent to webpage for decryption.

$$\begin{tikzcd}
\text{NFC} & \text{App} \arrow[l, "Pub_J"'] & \text{Webpage} \arrow[l, "{Pub_J, Sig_M, Pub_M, Sig_T}"'] & \text{Merchant} \arrow[l, "{Pub_J, Sig_M, Pub_M, Sig_T}"'] \\
\text{NFC} \arrow[r, "{c1, c2}"] & \text{App} \arrow[r, "{c1, c2}"] & \text{Webpage} \arrow[r, "\text{credit card info}"] & \text{Merchant}
\end{tikzcd}$$

Fig. 3. Transportation of private-key $Priv_J$. The merchant only needs to transport it to the webpage, where there is decryption of c_1, c_2.

$$\begin{tikzcd}
\text{NFC} & \text{App} \arrow[l] & \text{Webpage} \arrow[l] & \text{Merchant} \arrow[l, "Priv_J"']
\end{tikzcd}$$

4.4 Customer

The customer can use a one-way, unkeyed hash function to verify the app. We define a one-way, unkeyed hash function $h(x)$, where x is the input message, such that the probability of $h(x) = h(x')$ is unlikely $h(\cdot)^{-1}$ [1]. The customer can download a copy of the app and compute the hash of the app with the posted hash on the Tap-Card-Pay website.[7] If the download is different from the official Tap-Card-Pay download, the computed hash should be different from the expected hash.

5 Transaction Order of Operations

We define the order of operations when securing the credit card information. We see in Figure 2 how the public-keys and digital signatures are transported. In Figure 3, we see where is $Priv_J$ located and where it is transported from.

1. Merchant generates a keypair $Pub_J, Priv_J$.
2. Merchant posts the JavaScript with $Pub_M, Pub_J, Priv_J, Sig_M, Sig_T$.
3. JavaScript calls the PHP script to insert $Pub_M, Pub_J, Sig_M, Sig_T$ into *localhost*[8] MySQL database.
4. Customer clicks on the webpage link to launch the app.
5. App launches and requests that the customer tap the credit card next to the NFC.
6. App gets the credit card information from the API.
7. App generates a random symmetric key K and encrypts the credit card information using a block cipher to produce ciphertext c_1.
8. App downloads and verifies $Pub_J, Pub_M, Sig_M, Sig_T$. If verification fails, stop and show error message.
9. App encrypts K using Pub_J to produce ciphertext c_2.

[7] http://www.tapcardpay.com/

[8] *localhost* translates to the IP address 127.0.1.1 on the smartphone.

10. App enters c_1, c_2 into MySQL database.
11. JavaScript calls PHP script to retrieve c_1, c_2 from MySQL database.
12. JavaScript decrypts c_1, c_2 and presents the credit card information on the form and browser.
13. Customer decides whether or not to submit the credit card information to complete the transaction.

6 How to Break Tap-Card-Pay

We describe some of the security attacks on the proposed application.

6.1 Browser Cache

If the merchant enables caching for the webpages on the customer's (client) browser, it is possible to read the private-key $Priv_J$ from the browser cache.

6.2 Repudiation

The merchant can setup a second merchant website that Tap-Card-Pay Systems did not authorize. If the merchant's public-key Pub_M is not compromised and signs the public-key Pub_J, the merchant can lie and repudiate the signature Sig_M so that transaction messages are sent to the unauthorized merchant website. All transactions can be denied by the merchant so to avoid service agreements with Tap-Card-Pay Systems by making fradulent claims.

The merchant A can freely distribute the keypair $Pub_M, Priv_M$ to another unauthorized merchant B. All transactions can also be denied by the merchant A. Pub_M can be used to certify Pub_J and all transactions can be denied. A can remain silent about the usage from and existence of B.

6.3 Compromised Extension

The adversary can decrypt the ciphertext using $Priv_{J_1}$ if $Priv_{J_n} = ... = Priv_{J_2} = Priv_{J_1}$. This can happen if the pseudorandom key generation is corrupt.

6.4 Interception

Before the Tap-Card-Pay app gets the NFC data, the adversary can get the data. There is no restriction on who can read from the operating system API.

6.5 Negilence

The customer can accidentally scan the wrong credit card and submit wrong information to the merchant. The customer can blame the Tap-Card-Pay app for the bad transaction and demand compensation for damages.

6.6 Faulty Hardware and Operating System

Flash memory is nonvolatile memory that can be erased and reprogrammed and and many flash failure mechanisms worsen with cycling [3]. If the hardware or operating system fails to erase and save encrypted credit card information, wrong credit card information can be processed and submitted. The customer can blame the Tap-Card-Pay app for the bad transaction and demand compensation for damages.

In addition, when the app attempts to securely erase keys in memory, mechanisms used to check the wipe could fail. This leaves the keys in plaintext.

6.7 Replay

The adversary can block the operating system API call for NFC data and replay old credit card information to the Tap-Card-Pay app. The customer can blame the Tap-Card-Pay app for the bad transaction and demand compensation for damages.

6.8 Man-in-the-Middle

The Tap-Card-Pay website can be phished and the hash of a trojan app can be posted. The user would download and execute the trojan that captures credit card information.

6.9 Compromised Certificate Authority

When the adversary compromises the Tap-Card-Pay private-key $Priv_T$, it is possible to sign and authorize any merchant. Tap-Card-Pay Systems would need to upload new a public-key Pub_T into the app; otherwise, the unauthorized merchants could produce transactions without permission from Tap-Card-Pay Systems.

6.10 Untrusted Keypair Generation

The limititation is that the customer should not be required to read and input additionally information other than tapping and scanning the credit card next to the NFC.

Keypair generation per transaction is a costly process so the merchant may be tempted to offload this work to a third party. When offloading the keypair generation $Pub_J, Priv_J$ to the third party, the merchant must present to the customer the fact that the credit card information is also accessible by the third party. With a copy of $Priv_J$, an entity can decrypt encrypted messages produced with Pub_J. The customer would need to provide consent. Without the consent, the customer can blame the merchant for the breach of privacy and demand compensation for damages.

6.11 Poor Quality Symmetric Keys

The smartphone operating environment may not provide good entropy sources for pseudorandom number generation when generating the symmetric keys. K_1, ..K_n may be predictable.

6.12 Cryptanalysis

Cryptanalysis is outside of the scope of this paper.

7 Conclusion

We presented a public-key authenication and encryption application that secure the transactions between the merchant webpages and customers. We described some of the attacks that can break this. It would be exciting to see performance tests, additional attacks, or cryptanalysis before Tap-Card-Pay enters world-wide use.

References

1. A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone, “Handbook of Applied Cryptography,” CRC Press, 1996.
2. Android. Near Field Communication. https://developer.android.com/guide/topics/connectivity/nfc/index.html, accessed 7 November 2014.
3. Y. Chen, “Flash Memory Reliability NEPP 2008 Task Final Report,” JPL Publication 09-9 3/09.