Michael Seltzer
COMP 116: Security
Final Paper

# Client Side Encryption in the Web Browser
## Mentor: Ming Chow

## Abstract

Web service providers generally look to encryption as a means of ensuring data privacy between user accounts. When explaining security measures, such providers are often describing the actions that are performed not on the clients device, but rather once the data has been transmitted to server. Connections to servers are encrypted so that communications are secured via SSL[1] which prevents interception via an unrelated third party. However once the data is received and then encrypted on the server, the visibility control is held by the service provider. Not only can this allow for malicious employees to potentially access data, but when legally compelled, web services may be required to disclose personal information. Client side encryption aims to allows users to encrypt data before being sent to any external site so that they are the sole deciders of who can view personal, private information. In the web browser, client side encryption has not been widely used for a number of fundamentally challenging issues. Primarily, these issues lie with the problems of implementing client side encryption in many data heavy services and the difficulty in ensuring data has been securely encrypted. Users currently have little way to verify proper encryption in the browser - rather they must simply trust that website they visit provides adequate protections. In this paper an example of client side encryption using a javascript library will be used to show the possibilities of using the technology as well as the ways that an implementation can be manipulated and attacked. In addition a popular Android application, PushBullet which transmits information (such as pictures and SMS text messages) to a web interface will be evaluated for the use of end to end javascript client side encryption.

# Table of Contents

1 Introduction 4

2.1 To the Community: Current Encryption Practices 4

2.2 To the Community: Why Client Side? 5

3.1 Client Side Encryption Demonstration 5

3.2 Demonstration of Javascript Based Client Side Encryption Attacks 7

4 Problems Presented by Client Side Encryption 8

5. Case Study: PushBullet (End to End Encryption in a Service) 10

6. Conclusion and Future Research 12

References 13

## 1 Introduction

Although server side encryption provides an important level of security that hides data from most external bodies, it fundamentally cannot provide data privacy from the server itself. Most implementations of data security today involve trusting a server to encrypt data with their own encryption technologies (with no technical details confirming on the users side what, if any, encryption has occurred). Although server side encryption may be able to stop malicious activity, encryption keys must be stored on the server which means it is technically feasible for an employee or an external legal force to access "private" information. The intent of this paper is not to debate the legal ramifications (or how client side encryption may be used for nefarious purposes), but rather to discuss the feasibility of using javascript based client side encryption for the web to allow users to be the singular decider of who can view their private information. The reasons for considering client side encryption will first be considered, followed by a web based javascript demo of how this could be accomplished (with source code included on GitHub) and potential vulnerabilities. In addition a popular service that uses javascript based encryption will be evaluated as a case study.

## 2.1 To the Community: Current Encryption Practices

Server side encryption puts the server owner in charge of who can access data uploaded to a site. With recent articles[2] indicating that government authorities in the US want (or may already have) "backdoors" or live connection into mass data of popular online services, individual security and visibility choice is more important than ever before. Many companies use server side encryption which, while relatively secure in terms of external attacks, does not prevent all parties from viewing private data. Under current laws in many nations including the US (as shown by many transparency reports from companies such as Google[3]), a company can be legally required to turn over user data to further a legal proceeding.

## 2.2 To the Community: Why Client Side?

Client side encryption aims to hide data from all but the master key holder. By performing encryption at the client side, plaintext information is never transmitted outside of the users environment. Figure 1 below shows an example process of client side encryption at a high level. The client first downloads software to encrypt on the users device and then transmits the encrypted data to be stored on the server. To the server, the encrypted is data is a black box - that is it has no access to the contents or any way of manipulating the information inside.

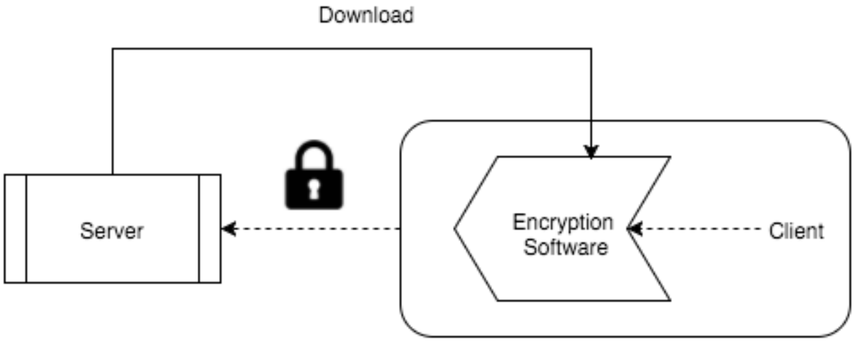


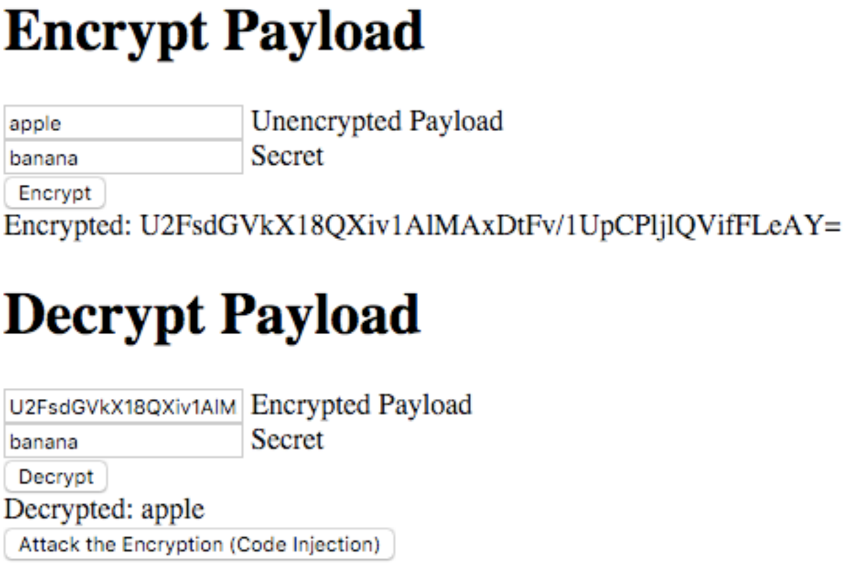
Figure 1. Client Side Encryption (High Level Overview)

## 3.1 Client Side Encryption Demonstration

In order to demonstrate client side encryption (and the risks) a demo application is available at http://mseltzer94.github.io/clientSideEncrypterDemo/ through the GitHub repository https://github.com/mseltzer94/clientSideEncrypterDemo. As shown in figure 2, the demo displays an option to take in two required fields to encrypt a string/payload using a secret passphrase. Clicking the “Encrypt” button will use a JavaScript based AES library[4] to encrypt the data which can then be decrypted. AES is the advanced encryption standard chosen by the National Institute of Standard and Technology (NIST), which uses the Rijndael cipher[5]. The method implements a symmetric algorithm meaning a single key is used for both encryption and decryption.

AES has been shown to be safe and the only vulnerabilities that have been demonstrated are so called side-channel attacks where an external detail about the processing of the algorithm allows an attacker to better crack the encryption. For example, the power consumption used during the cipher operation in AES has been shown to be indicative of the data being encrypted.[6] Although attacks like this are important in mission critical machines, AES is considered secure and side-channel exploits are extremely difficult and require access to the machine performing the encryption (on the client side for this case).

For the purposes of this demonstration it is assumed that the client computer does not contain malware or software that is recording keystrokes (which would negate any type of encryption be it server or client side). Rather, this demo shows that all encryption is performed on client side - the resulting encrypted output could be then sent to a server (which has no knowledge of the secret key). The server can then send back the encrypted payload which can only be decrypted using the secret key (which is only known to the client side). It should be noted that this example purely to show data hiding with access for a single key holder - it does not illustrate the sharing of encrypted data (trusted peer to trusted peer) which could be similarly implemented in JavaScript using a public and private key through technologies such as PGP (Pretty Good Privacy)[7].



Encrypt Payload

apple Unencrypted Payload
banana Secret

Encrypted: U2FsdGVkX18QXiv1AlMAxDtFv/1UpCPljlQVifFLeAY=

Decrypt Payload

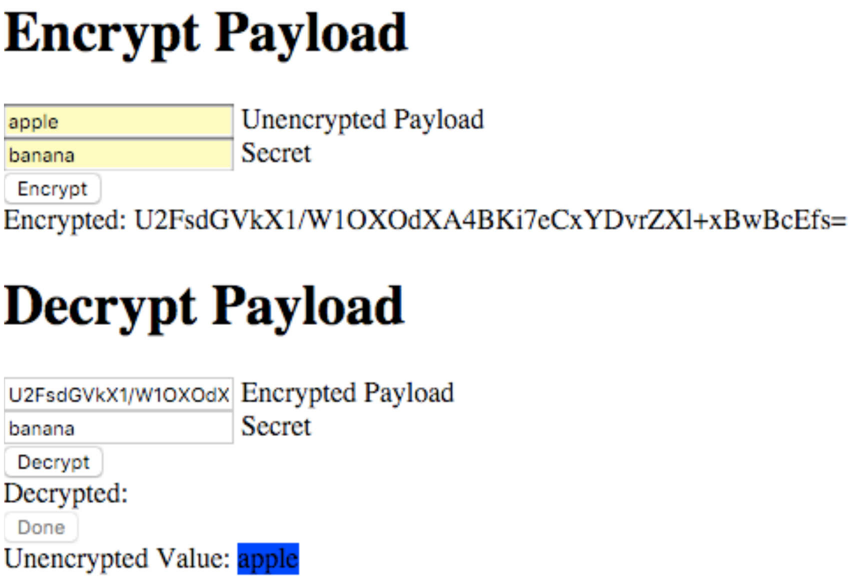
U2FsdGVkX18QXiv1AlM Encrypted Payload
banana Secret

Decrypted: apple

Figure 2. Demonstration of Client-Side Encryption using AES

## 3.2 Demonstration of Javascript Based Client Side Encryption Attacks

One fundamental issue with javascript based encryption is highlighted with a potential attack that can be demonstrated by clicking the “Attack the Encryption (Code Injection)” button in the demonstration mentioned previously. This attack injects javascript code which attaches itself to the “Encrypt” button and extracts the unencrypted payload before it is even sent to the AES encryption library. When activated, the attacker will append the unencrypted payload to the end of the page as shown in figure 3.



Encrypt Payload

apple Unencrypted Payload
banana Secret

Encrypted: U2FsdGVkX1/W1OXOdXA4BKi7eCxYDvrZXl+xBwBcEfs=

Decrypt Payload

U2FsdGVkX1/W1OXOdX Encrypted Payload
banana Secret

Decrypted:

Unencrypted Value: apple

Figure 3. Attack Retrieving Plaintext Payload before Encryption

Although harmless in this demo, in a truly malicious attack, the injected javascript could perform any arbitrary action such as silently sending unencrypted payloads (which the user believes to be encrypted) to a third party site.  In terms of real usage, this attack represents a fundamental issue with client side javascript encryption. The code injection that occurs in the attack is representative of what could happen if the server was taken over by a third party or a man in the middle attack occurred. The fundamental issue is that the client downloads the source code and in doing so trusts that the data never leaves the clients environment unencrypted. This form of code injection is further discussed below and described in figure 4.

## 4 Problems Presented by Client Side Encryption

Although client side encryption may have the potential to democratize security visibility and allow users to control who can view their private information, the difficulties of creating a truly random encryption cipher, possibility for malicious code injection, data access and data loss limit the ability for widespread adoption.  Firstly the AES library used in the demo does not contain a truly random number generator. Although AES is currently considered secure, in theory it is crackable[8]. In theory AES has been cracked[9], however performing such an attack is impractical in terms of time complexity. As with any encryption, AES encryption is safe for now but will eventually be broken as technology advances.

Another technical hurdle is that any code downloaded from a server is only safe if the transit lines are clear. If the server was compromised, the encryption that is fundamental to the service could be modified and previously encrypted data could become visible to malicious sources. Figure 4 below shows how an attack could be carried out, whereby the client side encryption code that is stored on the server is replaced with malicious code that does not in fact perform any encryption. When the user connects to the server and downloads the new code, they are unaware that the server side code has been replaced and that their data is now being sent in the clear.

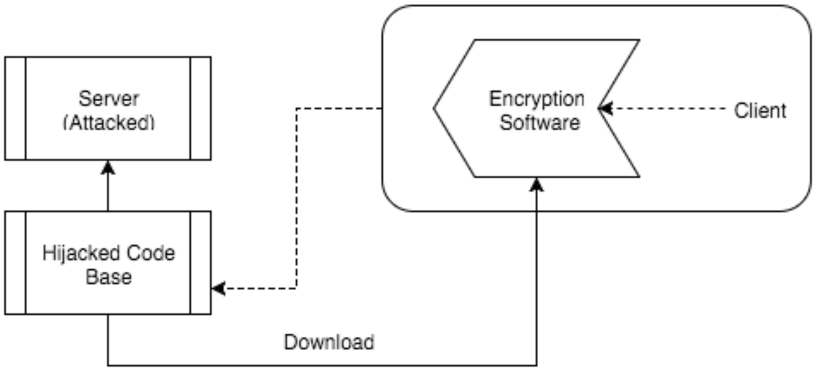


Figure 4.  Attacked Client Side Encryption (server side code injection)

The damages presented by a compromised server codebase could be mitigated using several techniques. Several research groups have created solutions[10] that require downloading add ons for encryption and decryption (such that they are static and trusted). The benefit of this strategy is that encryption (and the encryption strategy) is entirely separated from the server. For example, a group from MIT has built a Mozilla Firefox add-on that creates a sandbox where users can input private information (dependent on websites using their <encrypted> html tag)[11]. The issue becomes that users who don't use the encryption software will see encrypted "garbage" data on websites they access and will not be able to read information without the add-on. Although in corporate systems (where group policies are applied) this scheme may be possible, it is not a user friendly solution in general. Implementing client side encryption through javascript truly requires an additional layer - a trusted middleman - to ensure that the multitudes of servers the average person accesses everyday are not required to be instantly trusted by users. By performing checks and monitoring server code changes, the middleman could alert users to potential threats (and compromised encryption rules) preventing users from transmitting unencrypted information. Without a trusted middleman, javascript cryptography is susceptible to attacks as there is little way for a user to verify that their private information has been encrypted and is never sent in the clear.

One of the fundamental issues that prevents adoption of client side encryption is based on the most important tenant of the security strategy - that is, the server has no access to user data. While this may be feasible for data storage providers (which do view data as a black box), services that perform processing on the backend must push this responsibility to the client side device. Client side processing isn't feasible for services like Google or Facebook that provide a large amount of processing outside of the realm of the clients browser. Because of this fundamental hurdle many services encrypt on the backend and have access to the decryption key (and are therefore in charge of visibility control for the information held).

Another large issue with client side encryption is data loss. If a secret key is lost, data cannot be recovered (if the data is truly encrypted solely on the client side and the server never receives the key). Using techniques such as browser based local storage to remember secret keys may provide convenience, but also presents a security hole if left unsecured. For example, Apple uses a form of client side encryption for their person to person messaging platform, iMessage, and provides the user the opportunity to create a passcode which is used to decrypt the data where secret keys are stored[12]. In the case of iMessage, if a passcode is forgotten the information cannot be accessed (thus putting visibility control solely in the hands of the person who creates a passcode).

## 5. Case Study: PushBullet (End to End Encryption in a Service)

While javascript based client side encryption has not been used widely in commercial applications, there are some popular applications that include it to bolster security. One particular application that uses a client side encryption library is PushBullet[13]. PushBullet is a popular application that takes push notifications and SMS messages on an android phone and can forward them to a desktop using a browser extension. It allows a user to send files, messages from their desktop using a phone as the proxy for delivery. Because all notifications and messages are transmitted between a phone, PushBullet’s servers and a PC, security is a large concern. Because of this, the developers added an option for end to end encryption which allows a user to enter in a password and generate a symmetric key for all of the data that is sent and received. Their implementation uses a similar library to the one used in the demonstration above (using AES encryption similar to crypto-js). Although this encryption is very useful and provides an added layer of security, it by no means can quell fears of the developer (or a malicious third party) from injecting code to extract information before it is encrypted. The encryption works as follows - a user types in a password, clicks submit at which point a symmetric key is generated. The password then must be used on all devices in order for the messages (which are now encrypted) to be read on another device. Although the developer says that the password is never stored, there is no reasonable mechanism by which a user can deduce that this is actually true. Furthermore, if a legal force compelled the developers at PushBullet to hand over notifications from a specific user (or all of them), the technical capability to provide that kind of information is available. The chrome extension that is used to provide notifications and access to PushBullet can be updated, or worse yet the script that is performing encryption on a users password input can be fetched remotely (meaning it is always up to date). In fact, the developer has in the past pushed multiple updates in a day without any indication of changes to the user. Figure 5 below shows a modified version of PushBullet's javascript code that extracts the password input from a user form (and send it to be encrypted). The line highlighted in red shows an additional line that could be added so that the secret encryption passphrase could be sent to an external server (all other code is exactly as it appears in the current chrome extension code base, version 298).

```
save.onclick = function() {
	pb.e2e.setPassword(input.value);
};

sendToRemote(input.value);
```

Figure 5. Code sample from options.js (line 454) in PushBullet Chrome Extension code base (version 298)[14], red highlighted line is an additional line that could be used to perform extraction of the password before it is encrypted.

Even though the developer set out to create a secure environment where all data is encrypted without the servers knowledge, the reality is that because the developer is the one who provides this capability and ensures that the encryption occurs, users cannot have certainty that their data is inaccessible once it leaves their device. The problem in this situation is not specific to this case or developer but rather a problem for the system as it is setup right now - the server is the single identity that provides updatable code for encrypting and decrypting data and thus has control over the flow of information (be it for good or nefarious purposes).

## 6. Conclusion and Future Research

Although client side javascript encryption in theory could allow for user controlled security, the current barriers to implementation make it difficult to use in real world applications in the browser. Even with libraries that implement javascript versions of AES, a standard that is widely accepted for securing sensitive information, the potential for server code injection and the difficulties of creating a useful backend server that has no access to data are hard to overlook. Services that process data that is uploaded to their server cannot use client side encryption for much of the data that is transmitted as the server has no access to data once encrypted on the clients environment. Client side javascript, ironically, requires a user to trust the server in order to download code that could silently extract personal information. Unfortunately it is the flexibility and mutability of downloadable, constantly updating javascript itself that makes client side browser encryption susceptible to server code injection. Javascript based encryption provides opportunities to secure a users information before any external party could potentially access it but in order for services to implement the technology more work must be done to ensure that the encryption methods used cannot be circumvented by a malicious third party or through means of altering the server itself (legal or otherwise).

## References

[1] Digicert.com, 'What Is SSL (Secure Sockets Layer)? | DigiCert.com', 2015. [Online]. Available: https://www.digicert.com/ssl.htm.

[2] K. Zetter, 'NSA Boss: Encrypted Software Needs Government Backdoors', WIRED, 2015. [Online]. Available: http://www.wired.com/2015/09/nsa-boss-encrypted-software-needs-government-backdoors/.

[3]Google.com, 'Legal process – Google Transparency Report', 2015. [Online]. Available: https://www.google.com/transparencyreport/userdatarequests/legalprocess/.

[4] crypto-js', 2014. [Online]. Available: https://code.google.com/p/crypto-js/.

[5] A. Bogdanov, et al. Microsoft Research. Available: http://research.microsoft.com/en-us/projects/cryptanalysis/aesbc.pdf

[6] K. Meritt, et al. Differential Power Analysis attacks on AES, RIT. Available: http://people.rit.edu/kjm5923/DPA_attacks_on_AES.pdf

[7] P.  Zimmerman, 'Why I Wrote PGP'. 1991. [Online]. Available: https://www.philzimmermann.com/EN/essays/WhyIWrotePGP.html.

[8] BetaNews, 'Is AES encryption crackable?', 2009. [Online]. Available: http://betanews.com/2009/11/05/is-aes-encryption-crackable/.

[9] A. Biryukov, et al. Key Recovery Attacks of Practical Complexity on AES Variants With Up To 10 Rounds. Available: http://eprint.iacr.org/2009/374.pdf

[10] C. Velazco, 'Social Fortress Is A Simple (But Powerful) Skeleton Key/Data Security Service For Your Digital Life', TechCrunch, 2012. [Online]. Available: http://techcrunch.com/2012/09/10/social-fortress-is-a-simple-but-powerful-skeleton-keydata-security-service-for-your-digital-life/.

[11] S. Allen, et al. SecureForm: Secure Client-Side Browser Encryption. Available: http://css.csail.mit.edu/6.858/2013/projects/steb-praynaa-emilyrsu-nashah.pdf

[12] Support.apple.com, 'iCloud security and privacy overview', 2015. [Online]. Available: https://support.apple.com/en-us/HT202303.

[13] R. Oldenburg, 'End-To-End Encryption', PushBullet Blog, 2015. [Online]. Available: https://blog.pushbullet.com/2015/08/11/end-to-end-encryption/.

[14] Chrome Extensions, 'Pushbullet', 2015. [Online]. Available: https://chrome.google.com/webstore/detail/pushbullet/chlffgpmiacpedhhbkiomidkjlcfhogd?hl=en.