October 20, Mylar

Scribe: Qi Zhong

October 26, 2015

1 Introduction

Using a web application for confidential data requires the user to trust the server to protect the data from unauthorized disclosures. However, this trust at a lot of times are unfounded. With cloud outsourcing popular these days, admins at the cloud platform would have full access to the data. Hackers and the goverment may also gain access to the server. Mylar is web application platform designed to protect the data even if the attacker has full access to the data. The main idea behind Mylar is that only encrypted data are handled by the server. The server would not have the keys needed to decrypt the data so confidentiality is guaranteed.

2 Strawman

The simplest idea is that each user can have a unique key and all the files of that user is encrypted with that key.

2.1 Challenges

2.1.1 Functionality

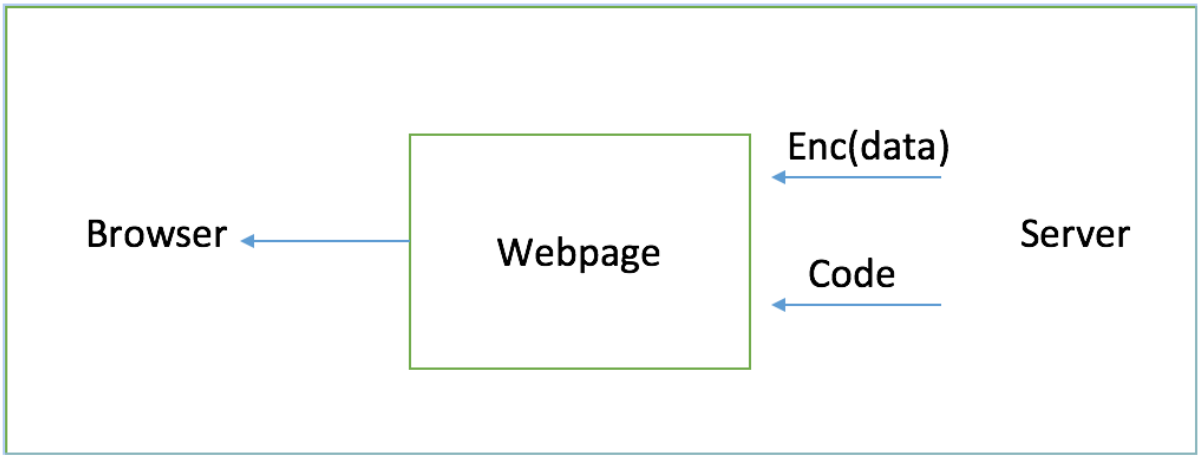
In this design, users would not be able to share data with each other. Since all the data are encrypted, the server also would not be able to format the input or provide basic functionalities like search. Even building a simple chat room application would be impossible now, because the two users can not see each other's messages. The server also cannot format or search the messages of the users.

2.1.2 Security

This scheme is still not safe because the server can serve malicious JavaScripts to the clients. It can use these scripts to still the key of the clients. Since the server serves all the HTML and script files, the client cannot verify the validity of them.

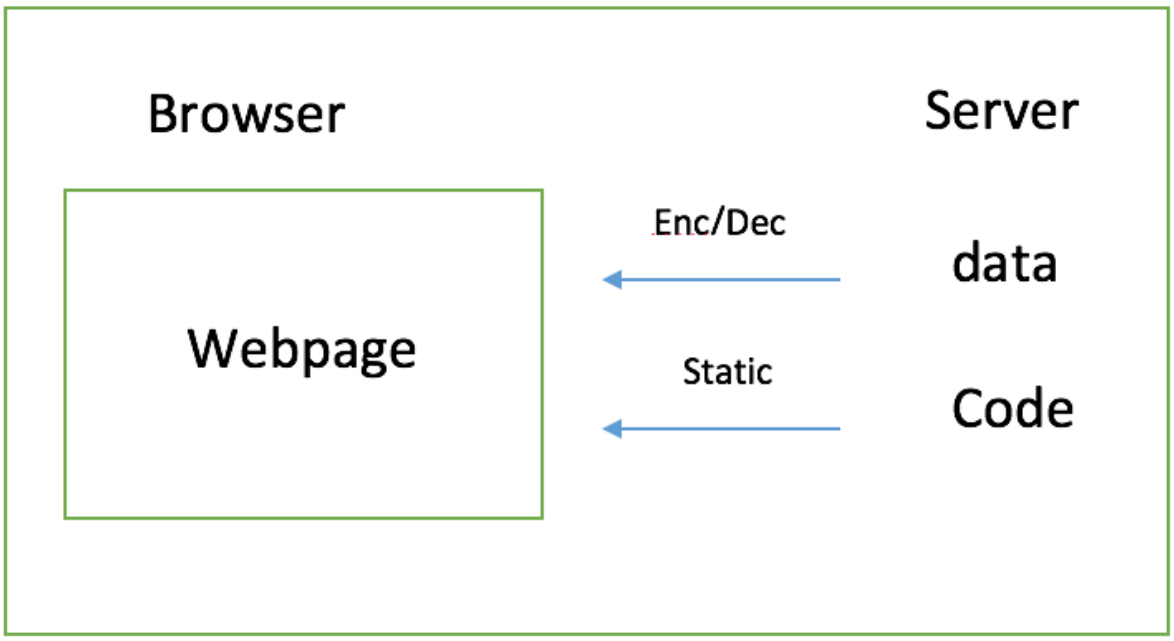
3 Framework structure

3.1 Django



This is the design of frameworks like django. The server create the webpage dynamically and sends it to the browser. With this kinds of frameworks, the server would need to manipulate encryped data, which is not possible. Also, since it is generated dynamically, it would be difficult for the client to check that the code is correct.

3.2 Mylar



In this design, all the encryption and decryption are done in the browser side. The code is also static. This way, the client can easily verify the code using the developer's signature.

4 Details

4.1 Access Graph

This specifies what principal has access what principal. It accepts commands like $createPrincipal(work)$, $addAccess(alice, bob)$

4.2 Sharing

For sharing, Mylar uses keys encryped in another key. For example, if Alice has access to the principal work, than her key to it would be $\{K_w\}_{K_a}$. This is work's key encrypted in her key. If she wants to share the access of chatroom work to Bob, she can re-encrypte the key to $\{K_w\}_{K_b}$ and send it to Bob.

4.3 Certificate Graph

The server can still lie during key distribution. For example, the server can send the key to the wrong chatroom to Bob and say that this is send by Alice. Then Bob would be talking to the wrong person without knowing it. So we need certificates to verify the idendity of keys. The certificate would contain the username of the person who signed the key and the certificate itself is signed by

the IDP.

4.4 DB

We also need a secure database to store all these sensitive information. The data base would contains the following information:

1. encryted key of access graph
2. encrypted data
3. certificates

4.5 Search

Mylar also supports doing searches in the server. The is achieved because it created a scheme where it can search across documents encrypted in different keys directly.

5 Presentations

5.1 ShadowCrypt

Unlike other designs where the application decides on what is encrypted when sending to the server, ShadowCrypt is created to allow the users themselves to make this choice. ShadowCrypt is a browser plug-in that would intercept all user input to textfields. Before putting them into the actual textfield of the site, all the text would be encrypted first. The site would only see the encrypted version of the input. When the user receives data from the server, all the encrypted data would be automatically decrypted and rendered correctly. The user would not notice that all his input has gone through both encryption and decryption

5.2 Hails

In cloud platforms, the platform sometimes needs to give access to its data with to an app. The usage of these data are usually controlled by a privacy policy, but it is hard to ensure that the app actually follows the terms of the policy. Hails is created to handle this problem. Hails is created on top of the traditional MVC model, which contains the modules model, view and controller. It adds a policy module to the MVC that controls the flow of data inside the app. For example, if the app tries to send the data to another server and it is forbidden by the policy, then the policy module would abort this action. This way, the terms of privacy policies can be enforced.