SecureWeb: Protecting Sensitive Information Through the Web Browser Extension with a Security Token

Shuang Liang, Yue Zhang, Bo Li, Xiaojie Guo, Chunfu Jia, and Zheli Liu*

Abstract: The leakage of sensitive data occurs on a large scale and with increasingly serious impact. It may cause privacy disclosure or even property damage. Password leakage is one of the fundamental reasons for information leakage, and its importance is must be emphasized because users are likely to use the same passwords for different Web application accounts. Existing approaches use a password manager and encrypted Web application to protect passwords and other sensitive data; however, they may be compromised or lack accessibility. The paper presents SecureWeb, which is a secure, practical, and user-controllable framework for mitigating the leakage of sensitive data. SecureWeb protects users’ passwords and aims to provide a unified protection solution to diverse sensitive data. The efficiency of the developed schemes is demonstrated and the results indicate that it has a low overhead and are of practical use.

Key words: password manager; data privacy; format-preserving encryption; Shadow Document Object Model (DOM)

1 Introduction

With the development of Web application systems, an increasing amount of user data is stored on the servers of online service providers. At the same time, the impact of sensitive information leakage is growing.

Many financial, online payment, and other applications are based on the Web where the user’s password, credit card number, and other information will become the main target of the attacker. There are various means for attackers to steal the sensitive data, such as compromising the Web application server, sniffing the network, or injecting malicious client-side code. Sensitive data leakage results in significant damage.

1.1 Motivation

The best way to protect sensitive data is to let its owner to encrypt it and be the only one who can decrypt it correctly. By giving users full control of their information, the risk caused by a compromised online server can be reduced. Many studies have focused on this idea, but all face the same challenge, which is key management. Some of the existing methods store the encryption key on the user’s personal computer, but an issue occurs when the same user wants to use another computer. Aiming to increase flexibility, other methods choose to store the key on online server, which is contrary to their original intention. What is more, the encryption and decryption processes are normally executed by Web browsers, which are undoubtedly less secure than executing the process by a stub program on
Common experience informs us that the password is the most valuable and vulnerable aspect of sensitive data. Until now, the password remains as the key component of most online authentication systems and a password leak directly threatens users. To enhance password security, the user should use unique and strong passwords for different accounts\textsuperscript{[2–4]}. However, it greatly increases the memory burden. As a result, many users choose to use same passwords on different websites, which is risky. This is because a hacker who has conquered a website also has the correct credentials for other websites.

This paper aims to protect the security of sensitive data, especially the security of passwords. The goal is to give users complete control of their personal data while maintaining accessibility. The confidentiality of the sensitive data should be guaranteed by the cryptographic operations executed on the user’s local computer, which can prevent the leakage of data from the computing environment.

1.2 Contribution

Based on these motivations, the SecureWeb framework is proposed, which is designed to encrypt sensitive data under the control of its owners. It uses a U-disk as the security token to authenticate users and enhance security. The key is stored in the U-disk rather than the local computer or the online server. Users hold the key so that they can control their own data. Choosing U-disk as the security token is a solution that maintains both the security and accessibility of the data. It is inexpensive and portable, so users can still read their data even if they change computers. In addition, SecureWeb builds a secure environment through the browser extension. What’s more, it implements the encryption and the decryption process by a stub program on the user’s local computer.

To protect the security of passwords, SecurePWD (shorted for SecurePassword) was implemented based on the SecureWeb framework. Password managers can manage passwords for users, but none of the current products can provide a secure input environment that prevents a malicious client-side code from capturing user inputs. Fortunately, the new technique, Shadow Document Object Model (DOM), can build a secure input environment by isolating application DOM. One of the best solutions is to use a password manager combined with Shadow DOM to achieve security and also ensure availability. This approach is unique. The SecurePWD can construct the secure environment by applying the Shadow DOM. With SecurePWD, users can retrieve different passwords for different Web sites on the basis of the unique master password.

To protect the sensitive data, SecureData is implemented and is based on the SecureWeb framework, too. It can be seen as an extension of SecurePWD. Besides the protection of users’ passwords, SecureData gives a common implementation to put users in control of their sensitive data. SecureData provides a unified protection solution to diverse sensitive data.

2 Related Work

2.1 Encrypted Web application

Web applications usually store sensitive data generated by users. Users may worry about the personal data privacy or information privacy due to compromised or curious Web applications. The best solution is to store the encrypted data such that it can only be decrypted by the owner.

Recently, studies\textsuperscript{[5–11]} have focused on data security and privacy preservation, and have proposed many useful solutions. They show that there are three chokepoints that may leak a user’s sensitive data in Web applications. As shown in Fig. 1, the data can be leaked: (1) from the database of the application server in chokepoint (a); (2) when transferred over the Internet in chokepoint (b); and (3) by malicious client-side code in chokepoint (c).

To encrypt Web application data, many proposals adopt the approach of encrypting the data at one of the three chokepoints. It is clear that a higher security level will be achieved when the chokepoint is further to the right. However, it is a huge challenge to make the encryption controlled by the user while maintaining transparency.

ShadowCrypt\textsuperscript{[8]}, which is proposed in ACM Conference on Computer and Communications Security (CCS), enables users to encrypt the sensitive data in chokepoint (c). It stores the encryption

Fig. 1 Chokepoints for data leak in Web applications.
key on the user’s computer and performs the encryption and decryption operations in the browser extension. However, users cannot control their encryption key directly by themselves. In addition, once the user changes a computer, the encrypted data cannot be decrypted correctly. In other words, it lacks accessibility.

2.2 Password manager

Data leak, especially password leak, is a huge risk to Web application users. To enhance password security, only unique and strong passwords should be utilized; nevertheless, this greatly increases a burden on the user’s memory. The password manager can help users to store and organize their passwords. Most of them are based on the following methods.

- **Web-based password manager.** This stores users’ passwords on the online service provider’s servers. LastPass\(^\text{[12]}\) and RoboForm\(^\text{[13]}\), which are based on this method, encrypt a user’s password database using the user’s master password. But if the master password is not a very strong one, or the user uses the same password on the other sites, the user risks losing all of their secrets. Many studies\(^\text{[14–16]}\) show that most Web-based password managers have a large attack surface. Recently, a group of security experts from Team Security Is Key (SIK) of the Fraunhofer Institute for Secure Information Technology in Germany found that many Web-based password managers, such as LastPass, hardcode the master password in its application and can therefore leak users’ privacy data\(^\text{[17]}\). What’s more, the Web-based password manager stores credentials on the cloud that can easily be a target for attackers. If the service provider’s server is intruded, the user will face a huge risk.

- **Local password manager.** This works similarly to the online Web-based password manager. The local password manager just saves the encrypted password database on user’s computer rather than on the Web. It avoids the security weakness of storing online. Popular ones include 1password\(^\text{[18]}\) and the desktop version of RoboForm\(^\text{[13]}\). Team SIK also found that 1password is vulnerable\(^\text{[17]}\). Besides, the biggest weakness of the local password managers is the lack of accessibility.

- **Security token password manager.** It manages user’s passwords with a security token. It is seen as the best way to authenticate users. The data stored in the token is usually encrypted to prevent it from unauthorized probing and reading. However, the main disadvantage of it is the cost of ownership. Users may not use the security token password managers due to this reason.

Table 1 summarizes the characteristics of the above password managers. As can be seen, none of them reach the ideal expectation. A good password manager must have a small attack surface, high accessibility, and low cost.

However, using a password manager to organize passwords does not mean that the passwords are secure. Some studies\(^\text{[19–21]}\) have proved that normal users’ passwords are far from secure against attacks. More specifically, based on some known information of users, attackers can guess the password with a very high probability\(^\text{[21]}\). Therefore, only the random and unique password is secure.

3 SecureWeb Architecture

To maximize the security of personal sensitive data, SecureWeb aims to encrypt users’ data at chokepoint (c). In order to give users complete control over their personal sensitive data, SecureWeb stores the encryption key in a U-disk that is convenient to carry around and not expensive. What’s more, instead of performing the encryption and decryption in the Web browser, SecureWeb executes these operations in a stub program, which further enhances the security.

3.1 System module

As shown in Fig. 2, SecureWeb contains three modules (i.e., the secure environment module, the encryption module, and the key management module) and a communication protocol.

**Secure environment module.** This implements the identification of sensitive fields and constructs a secure environment by isolating the application DOM. This module is implemented by the browser extension.

| Password manager                  | Attack surface | Accessibility | Cost    |
|-----------------------------------|----------------|---------------|---------|
| Web-based password manager        | Large          | High          | Low     |
| Local password manager            | Small          | Low           | Low     |
| Security token password Manager   | Small          | Medium/high   | Medium/high |
Shuang Liang et al.: SecureWeb: Protecting Sensitive Information Through the Web Browser Extension with a...

Fig. 2 Modules of SecureWeb.

**Encryption module.** This uses the results of the secure environment module as its input, processes (i.e., encrypts or decrypts) it based on different website domains and a security token, and updates the new data to the original fields. This module is implemented by the stub program.

**Key management module.** This maintains a key file in a security token that is the U-disk to authenticate users.

**Communication protocol.** This is the protocol between the secure environment module and the encryption module. To guarantee the secure transmission of the users’ data, the Diffie-Hellman (DH) key exchange protocol is used.

The key challenge is how to process the sensitive data. It all depends on whether the encrypted data need to be decrypted or not, as will be described in Section 3.3.5.

### 3.2 Data flow

The data flow of SecureWeb will be briefly described. The steps of the data flow go from 1 to 6, as shown in Fig. 3.

Notice that SecureWeb needs an initialization when it is first used. This is drawn as step 0. In Fig. 3, the solid line presents the data flow that interacts directly with the sensitive data, whereas the dashed lines do not. What’s more, to simplify the data flow, the communication protocol was omitted here.

**Step 0.** Initialization of key management module. SecureWeb first generates the user key file which will be provided to process the sensitive data. The key management module dynamically obtains the physical serial number of the specific security token, i.e., the U-disk. Then it uses this unique physical serial number to encrypt the key file.

**Step 1.** Identify the sensitive field and domain. SecureWeb then identifies the sensitive fields in the application code and builds a secure environment for these fields. The cleartext of sensitive data interacts directly with the secure environment rather than the original Web page. SecureWeb will also identify the domain name of the current Web page.

**Step 2.** Send the sensitive data and domain. SecureWeb sends the sensitive data and the domain name of the current Web site from the secure environment module to the encryption module. The secure environment module also establishes a corresponding callback function to handle the data sent back from the encryption module.

**Step 3.** Process the sensitive data. SecureWeb obtains the U-disk physical serial number and uses it to decrypt the key file. After decryption, SecureWeb gets the content of the key file. Then, it conducts the key dispersion and encrypts the domain name by an encryption key read from the key file to get the session key. Finally, it processes the sensitive data received from the secure environment using the session key.

**Step 4.** Update the key file. SecureWeb updates the key file with the time of last access and the frequency of accesses.

**Step 5.** Send the processed data back. SecureWeb sends the processed sensitive data back to the secure environment module.

**Step 6.** Update the original Web page. The callback function in the secure environment module receives the processed data. Then SecureWeb updates the original Web page with it.

### 3.3 Goals

The goals are to construct a sensitive data protection mechanism with users’ complete control. In this section, the security and usability goals are presented. The non-goals that were not considered in SecureWeb are also presented.

#### 3.3.1 Security goals

The security goals of SecureWeb are as follows:

**Avoid the compromised online storage device.** SecureWeb aims to put users in control of their data and mitigate the privacy diffusion problem. The online service provider’s server cannot read the cleartext of users’ sensitive data. Therefore, SecureWeb can avoid the risk of online storage devices being compromised.

**Provide secure environment.** SecureWeb aims to ensure the security of sensitive data. To prevent the
sensitive data from being stolen by the malicious client-side code, SecureWeb provides a secure isolated environment for users.

**Ensure the security of the password.** SecureWeb aims to protect sensitive data, especially the password. It aims to set a unique password for each website. Leaks of the passwords from some weak websites will not open the gate for the attacker to enter the other websites.

### 3.3.2 Usability goals

The usability goals of SecureWeb are as follows:

**Transparent to the Web application.** SecureWeb focuses on supporting applications transparently. It will not break the verification and other logic of the Web application.

**Accessible and inexpensive.** SecureWeb uses a security token to enhance the security for users. The security token should be inexpensive and easy to carry around. Besides, in contrast to the former method[8], users can read their data in any computer with the security token.

### 3.3.3 Non-goals

We do not aim to protect against the threats by losing the security token. We assume that the user can prevent the security token from being lost.

### 3.3.4 Security analysis

SecureWeb may have three threats, as shown in Fig. 4, and they are as follows:

1. Threats between the application code and the secure environment module;
2. Threats between the secure environment module and the encryption module;
3. Threats between the encryption module and the key management module.

**Claim 1** SecureWeb can resist attacks between the application code and the secure environment module.

**Proof** The malicious client-side code can steal user’s data if the secure environment module is not isolated from the application code. Fortunately, the new technique, named “Shadow DOM”, can isolate application DOM[18,22]. This ensures the security between the application code and secure environment module.

**Claim 2** SecureWeb can resist attacks between the secure environment module and the encryption module.

**Proof** Attackers can steal user’s data by hook function or other methods if the data is transferred in plaintext. SecureWeb uses DH key exchange protocol to prevent that exposure. Thus, the security between the secure environment module and the encryption module is guaranteed by our communication protocol.

**Claim 3** SecureWeb can resist attacks between the encryption module and the key management module.

**Proof** Attackers may steal the key file in order to decrypt user’s encrypted data. But the key file is encrypted by the physical serial number of the U-disk that is dynamically obtained by SecureWeb. Thus, this attack will be successful only when the attacker gets the security token (i.e., U-disk). In this paper, it is assumed users can prevent the security token from being stolen. Therefore, the security between the encryption module and the key management module is guaranteed.

### 3.3.5 Case study

As mentioned before, whether the encrypted data needs to be decrypted or not, determines how SecureWeb processes sensitive data. Two cases are considered, which are called SecurePWD and SecureData, respectively.

**SecurePWD.** The password is one of the most secret and sensitive pieces of data that needs to be unique and strong. One of the goals of SecureWeb is to protect the security of passwords. The ideal solution is to set a unique password for each website, but how to manage these passwords is troublesome for users. In this case, SecurePWD based on the SecureWeb framework is implemented. SecurePWD encrypts the master password that is input by users to get the login password based on the current website domain and the U-disk. SecurePWD generates different passwords for different websites. Whenever users need to login, SecurePWD will compute the current login password. Therefore, SecurePWD is unidirectional and does not need to process the decryption operation.

**SecureData.** SecureData is the general implementation of SecureWeb. It provides a unified protection solution to diverse sensitive data. It encrypts the sensitive data of users’ input. It differs from SecurePWD in that SecureData is bidirectional, i.e., it
needs to decrypt the ciphertext of the sensitive data in order to present to users. SecureData needs to identify the sensitive input widgets to encrypt the users’ input; what’s more, it also needs to identify the encrypted sensitive data to process the decryption.

4 SecurePWD Implementation

In this section, details about the implementations of SecurePWD are provided.

4.1 Secure environment module

4.1.1 Overview

The secure environment module, running as a browser extension, aims to provide a secure password input environment for the user and prevent the master password from being stolen by the malicious client-side code (such as a Cross Site Scripting (XSS) attack).

It first traverses the input elements in the application and finds the password input fields, then creates secure input environments for these fields, and finally sends the data (i.e., master password) obtained by the user’s input to the encryption module. It will also receive the encrypted data sent from the encryption module and then update the value into the original password input field of the webpage.

To achieve the isolation, the application DOM must be isolated. There are several ways \cite{8, 22} to isolate the application DOM, such as iframe, modifying the DOM Application Programming Interface (API), or Shadow DOM:

- Iframe. This uses a browser’s built-in frame and original isolation properties. However, this method may bring discord because it disregards the surrounding page’s text styling.
- Modifying the DOM API. One can override the input element to provide an isolated secure widget. But modifying the DOM API is a very complex and difficult job. Without proficiency in formalizing the DOM API and its semantics, this method seems hard to be realized.
- Shadow DOM. This is a new feature proposed in the HTML 5 specification. It allows the Web developers to encapsulate codes. It is flexible enough to use Shadow DOM to isolate Web application DOM. This will be described in more detail in the next section.

4.1.2 Shadow DOM

Shadow DOM provides encapsulation for DOM and Cascading Style Sheets (CSS) in a Web component. Shadow DOM makes them separate from the DOM of the main document. It can prevent outside JavaScript code from accessing the content in the shadow tree. Now, it has been updated from version v0 to version v1. The differences between the two versions are listed in Table 2 and are as follows.

1. Creating a shadow root. In Shadow DOM v0, creating a shadow root is executed by function `Element.createShadowRoot()` while in Shadow DOM v1, it is executed by function `Element.attachShadow()`. What’s more, a mode that is a string specifying the encapsulation mode for the shadow tree must be given as a parameter in that function.

2. Mode of the shadow root. There are two modes of the shadow root: open and closed. With the open mode, one can access the Shadow DOM via the `shadowRoot` property of the HTML element. With the closed mode, one cannot access anything inside the shadow tree, and `shadowRoot` will return null, i.e., the design goal of a closed mode is to disallow any access to a node in a closed shadow root from the outside world. Besides, in v0, a shadow root is always open; in v1, it can be set to open or closed.

3. Elements which can be shadow host. In Shadow DOM v0, every element can be a shadow host. In Shadow DOM v1, a limited number of elements can be a shadow host. For example, the input element can be a shadow host just in v0.

The nature of Shadow DOM composition is explained more specifically with an example in Fig. 5. A shadow tree is created with the HTML paragraph element whose ID equals to “1” and serves as the shadow host. A closed mode disallows any access to a node in a shadow root from the outside world. A new paragraph element is created and as the shadow host. When executing

| Table 2 Differences between Shadow DOM v0 and Shadow DOM v1. |
|---------------------------------------------------------------|
| **Difference**      | **Shadow DOM v0**     | **Shadow DOM v1**  |
| Creating a shadow root | `Element.createShadowRoot()` | `Element.attachShadow(mode: “closed” or “open”)` |
| Mode of the shadow root | Open | Open or closed |
| Elements which can be shadow host | Every element | Partial elements |
Fig. 5 The sample of creating a shadow tree.

document.getElementById("2") through the console, null will be returned. That is, the outside JavaScript code cannot acquire the content within Shadow DOM, which exemplifies the encapsulation of Shadow DOM.

The HTML page generated by this code will display “New world” instead of “Hello world!” because the shadow host will not be rendered.

4.1.3 Method details

To isolate the secure password input environment from the application DOM, the traditional way that is based on Shadow DOM v0 chooses the input element to host a shadow tree, as shown in Fig. 6a. The new input element in the shadow tree will be rendered in place of the shadow host (i.e., the original input element in the Web application code).

With the version upgrade of Shadow DOM, this method cannot work on most Web browsers. Because in Shadow DOM v1, only partial elements can be the shadow host, and the input element cannot be the shadow host anymore.

To take advantage of the flexibility and isolation of Shadow DOM, the limitations of the traditional method are made up. The process is described (shown in Fig. 6b) as follows.

To ensure that the cleartext by user’s input is contained in a shadow tree, one should choose a legal element to host the shadow tree. In this situation, as the input element cannot host a shadow tree, choose the div element to be the shadow host that is supported in Shadow DOM v1. What’s more, set the mode of shadow root closed in order to prevent any access to the node in a shadow root from the outside world. Then, to add it into the Web application DOM, insert it into the DOM tree as the brother node of the original password input element.

Now, a secure isolated input widget has been built. However, adding the new input element into the Web application will cause a collision in the display. Therefore, hide the original password input element by modifying the CSS code.

Notice that the secure environment module will listen on users’ keystroke events and transfer the master password to the encryption module. After being encrypted by the encryption module, the password will be updated to the original password input element. So that user interacts directly with the secure input environment and it will not badly influence the normal logic of the Web application. The sample of the whole process is shown in Fig. 7.

4.2 Encryption module

4.2.1 Overview

The encryption module, running as a stub program, aims to provide the encryption for the master password. This module receives the user’s master password and the current domain from the secure environment module and the key file from the key management module. Then, the encryption module dynamically obtains the physical serial number of the security token and decrypts the key file with it. After the decryption,
it will get the master key. SecurePWD uses this key to encrypt the master password and updates the secure environment module with the login password. Therefore, the login password depends on the master password and the domain of current Web site. The same master password with different domains will be generated for different login passwords. The same domain with the different master passwords will be generated to different login passwords, too.

### 4.2.2 Problem statement

However, a traditional encryption algorithm such as Advance Encryption Standard (AES) in this situation will lead to some problems. For example, ciphertext may break the application front-end and back-end verification because the format of ciphertext is different from the cleartext\(^{23, 24}\). To solve this problem, an encryption algorithm to generate ciphertext was chosen that falls in the same domain as the plaintext.

Format-Preserving Encryption (FPE)\(^{25–29}\) that has been proposed recently aims to solve that problem\(^{30, 31}\). The motivation of applying FPE comes from the problems associated with encryption for restricted field lengths or formats. The typical schemes of FPE are integer\(^{32}\), character data\(^{29}\), datetime\(^{25}\), etc.

### 4.2.3 Our encryption algorithm

In order to ensure the security of encryption and decryption, most FPE schemes combine cycle walking and an AES-based balanced feistel network\(^{33}\). However, in this situation, the master password must be encrypted to get different login passwords and it does not need to be decrypted, i.e., a unidirectional encryption algorithm that preserves the format is needed.

The encryption algorithm (as shown in Algorithm 1 and Fig. 8) is described as follows.

- **Key Dispersion.** The domain name of the current Web site was chosen as the dispersion factor that will be encrypted by the encryption key. The function \(Enc(Udisk\_key, domain, session\_key)\) where \(Enc\) refers to the AES encryption algorithm, and the session key refers to the result of key dispersion.

- **Format-preserving encryption.** After key dispersion, the session key is used to encrypt the master password. The AES encryption algorithm is selected. Then, the first \(N\) characters are chosen, where \(N\) is equal to the length of the master password. In order to preserve the format, a modulo operation is taken here to make sure the ciphertext falls in the same domain as the plaintext, i.e., after encryption, a capital letter is still a capital letter, a number is still a number, and so on.

**Security analysis.** The measure of a secure format-preserving encryption algorithm is whether an attacker can distinguish the format-preserving encryption from a pseudorandom permutation. The security goal here is Pseudorandom Permutation (PRP) security.

To describe the FPE algorithm and the pseudorandom permutation, a value was chosen from \([0, 1]\) randomly: \(b \leftarrow \{0, 1\}\). A permutation was chosen randomly: \(P \leftarrow \text{Perm}(-)\), where \(\text{Perm}(-)\) denotes the set of permutation.

Let \(A\) indicate an adversary with an oracle which is either an encryption oracle or a permutation oracle. Then the attacker, \(A\), can be seen as the adversary of format-preserving encryption \(Enc\). To give attacker adequate attack advantages, the attacker is allowed to perform an encryption query and get response from the encryption oracle \(Oracle(Encryption)\). The encryption oracle allows the attacker to choose arbitrary plaintext and obtain the corresponding ciphertext.

According to the attacker’s encryption query, based on the value of \(b\) chosen randomly, the encryption oracle decides whether to adopt pseudorandom permutation or encryption oracle to response. If \(b = 0\), then \(Oracle(Encryption)\) is responded by a permutation

---

**Algorithm 1** Format-preserving encryption

```
1: \(N\) is the length of the master password.
2: \(login\_password[N]\) is an array of length \(N\).
3: \(AES\_E\_\text{master}\_\text{password}(plaintext)\) is the AES encryption algorithm.
4: \(Modulo\) is the function of doing modulo operation.
5: session\_key \leftarrow AES\_E\_\text{domain}\_\text{session}\_\text{key}(domain)
6: login\_password\_tmp \leftarrow AES\_E\_\text{login}\_\text{password}(master\_password)
7: for each \(i \in [1, N]\) do
8:     login\_password[i] \leftarrow Modulo(login\_password\_tmp[i])
9: end for
10: return login\_password
```
oracle $P(\cdot)$, otherwise if $b = 1$, Oracle (Encryption) is responded by our format-preserving encryption $Enc$.

The attacker’s goal is to verify if it is an encryption oracle $Enc$ or a permutation oracle $P(\cdot)$ after a certain times of encryption queries. In other words, the attacker aims to get a determined value $b'$. If $b' = b$, the attacker wins the game, otherwise he/she fails. The advantage of the attacker $A$ can be described as $Adv_{PRP,Enc}^A = 2Pr[PRP^4_{Enc} \Rightarrow true] - 1$, where $Pr[PRP^4_{Enc} \Rightarrow true]$ is the probability of attacker to win the game.

The encryption oracle $Enc$ of the FPE algorithm is based on AES algorithm. Therefore, the attacker’s advantage is equivalent to $Adv_{PRP,Enc}^A = 2Pr[PRP^4_{AES} \Rightarrow true] - 1$. It is obvious that if the AES algorithm is a random permutation, the attacker cannot obtain adequate advantage. Therefore, the security of our format-preserving algorithm depends on the AES algorithm. As the AES algorithm is proved to reach the PRP security, therefore, our format-preserving algorithm reaches the PRP security.

### 4.3 Key management module

The key management module maintains a key file stored in the security token. When it is used for the first time, one needs to initialize the security token in order to generate a unique key file into the U-disk for the user. To prevent the key file from being copied to the other U-disks and revealing the encryption key, this file is encrypted by the physical serial number of the U-disk.

The format of the key file is shown in Fig. 9, which consists of an 80-bit time stamp, a 16-bit visit time, and a 256-bit key. The 80-bit time stamp and 16-bit visit time provide an access control for the key file in SecurePWD. The time stamp records the time of last access. The visit time records frequency of accesses. If the frequency exceeds the secure threshold, the access will be rejected. Therefore, the key management module makes the brute force attack impossible. The key used in AES-256 is initialized randomly based on the time seed.

### 5 SecureData Implementation

In this section, the details about the implementation of SecureData are given. The SecureData can be seen as the extension of SecurePWD. They have much in common and this section will focus on the differences between them.

#### 5.1 Input sensitive data

When a user inputs sensitive data, SecureData builds a secure environment that is similar to SecurePWD. Then, the secure environment module transfers the sensitive data to the encryption module that will do the encryption.

In SecurePWD, the encryption algorithm is format-preserving so that the format of the encrypted master password is the same as the original one. However, applying the format-preserving encryption algorithm in SecureData seems impossible. The sensitive data encrypted by SecurePWD need not to be decrypted, whereas the sensitive data encrypted by SecureData must be able to be decrypted so that users can use it. The encrypted data must be able to be identified; therefore, the format-preserving algorithm in SecurePWD is not applicable in this case.

The encryption module of SecurePWD contains two parts: the key dispersion process and the format-preserving encryption process. SecureData just selects the first part, i.e., the key dispersion. The encryption module of SecureData receives the domain name of the current website from the secure environment module and the encryption key from the key management module, then it encrypts the domain name by the encryption key to get the session key. Then SecureData uses AES encryption algorithm to encrypt the sensitive data with this session key.

The ciphertext seems different from the cleartext with the traditional block cipher algorithm due to the block length. However, it’s hard to distinguish by the computer programs. For this reason, SecureData adds a signature to identify the ciphertext. SecureData uses `==SecureDataStart==` and `==SecureDataEnd==` to mark the encrypted sensitive data. For example, `==SecureDataStart== ciphertext ==SecureDataEnd==` stands for the data which need to be decrypted.

#### 5.2 Output sensitive data

To make the encryption transparent to users, SecureData identifies the encrypted sensitive data and decrypts it when it needs to be rendered.

In order to find the encrypted sensitive data, SecureData searches the `==SecureDataStart==`
Shuang Liang et al.: SecureWeb: Protecting Sensitive Information Through the Web Browser Extension with a... 535

and ==SecureDataEnd== signatures. This identification is realized by the secure environment module. The string between the two signatures is identified as the ciphertext which needs to be decrypted. Then the secure environment module sends the ciphertext and the domain of current Web site to the encryption module.

The encryption module of SecureData receives the ciphertext and domain from the secure environment module and the encryption key from the key management module. It first computes the session key based on the domain name and encryption key. Then the encryption module decrypts the ciphertext with the session key.

After decryption, the encryption module sends the results to the secure environment module. The secure environment module builds a secure environment for it and renders it to the user.

6 Evaluation

Based on SecureWeb, two concrete implementations, SecurePWD and SecureData, are provided that have a lot in common. To evaluate them, two aspects were the focus: (1) the time cost of creating a secure environment and (2) the time cost of encryption.

The performance overhead was measured by the number of input elements of sensitive data. The test was conducted on an Intel Core i7-6700 3.41 GHz x 8 with 16 GB of Random Access Memory (RAM).

Time cost of creating secure environment. As both SecurePWD and SecureData need to create a secure environment, in this test Web pages with 1–100 input elements were established, and the result is shown in Fig. 10. As illustrated, the time cost for SecurePWD and SecureData to create secure environments increases with the increase of sensitive fields. This is because the increase in the number of elements will cause an increase in the time of operations.

The result shows that the cost of creating 10 secure input widgets is approximately 3 ms, and it takes less than 10 ms to create 100 secure input widgets. The time cost can be accepted by users. Notice that this test is far beyond the practical use for SecurePWD because the password input field in reality is usually one. The time cost to create a secure input environment for one input widget is approximately 2 ms, which is imperceptible to users. Above all, the time cost in this step can be ignored for the user experience.

Time cost of encryption. Time cost of encryption.

The time cost of encryption was first tested in SecurePWD by using different algorithms. The encryption module of SecurePWD encrypts the user’s master password received from the secure environment module. To estimate the effect of the encryption efficiency of the algorithm, two kinds of encryption schemes were chosen, namely the FFX (Format-preserving, Feistel-based encryption, and X is the parameter choice) mode of the Format-Preserving Encryption (FPE) algorithm and the AES algorithm. Six rounds of the format-preserving algorithm were used. To simulate the actual situation, the strings with a length from 1 to 16 were chosen because the length of login passwords, in reality, will generally not be longer than 16 characters. The execution time of the encryption functions were logged in the console.

The result of the test is illustrated in Fig. 11. It is clear that: (1) when FFX was used to encrypt strings with a length less than 16, the time cost is between 1.2 ms and 1.4 ms; (2) when AES was used to encrypt strings with a length less than 16, the time cost is about
0.2 ms; and (3) when the format-preserving algorithm was used to encrypt strings with a length less than 16, the time cost is approximate to 0.3 ms, which is little more than AES algorithm and far less than FFX algorithm. This is because FFX needs to do several rounds of cycle walking based on AES, while the algorithm created here does not.

Then, the time cost for SecureData was tested. The encryption algorithm in SecureData is similar to SecurePWD, wherein the AES encryption algorithm is chosen. The main difference between them is that SecurePWD just encrypts the data with a short length, whereas the SecureData needs to encrypt a longer message. Therefore, the time cost for SecureData to encrypt a message that has a length from 1 to 1000 characters was tested.

As illustrated in Fig. 12, the encryption takes under 2 ms for messages up to 100 characters long and 14 ms for messages up to 1000 characters long. Therefore, users will not notice the overhead.

The above experiments were based on the Google Chrome browser. The time interval between inputting the cleartext and receiving the encrypted data in the secure environment module was tested. Next, the time cost for encryption on different Web browsers was tested. The time cost for SecureData to encrypt messages with a length less than 1000 characters long on the 360 and Baidu browsers was tested. The result of the test is illustrated in Table 3. It was found that there are minor differences in the time cost on different Web browsers. However, the overhead is perfectly acceptable on all of these browsers.

Overall, the performance is reasonably fast and the cost can be completely accepted by the user.

7 Conclusion

This paper presented SecureWeb, a sensitive data protection mechanism. Based on a browser extension, SecureWeb provides a secure environment for user to interact with the sensitive data. It provides encryption and/or decryption in a stub program on users’ local computer to put users in the control of their sensitive data. What’s more, it chooses the U-disk as a security token that enhances the security and provides a high accessibility.

With SecureWeb, an application called SecurePWD was implemented for password protection, which is now an urgent need and in close correlation with user information security. In contrast to previous approaches, SecurePWD does not trust the security of password storage devices. SecurePWD does not store any passwords; instead, it dynamically generates a login password based on the master password and the domain of the current website. Besides, we also extensively apply our mechanism in the protection of diverse sensitive data, and implement SecureData.

The efficiency was evaluated and the experiments demonstrated that this new mechanism is of practical use and has a low overhead.

Acknowledgment

This work was supported by the National Key Basic Research Program of China (No. 2013CB834204), the National Natural Science Foundation of China (Nos. 61672300 and 61772291), the Natural Science Foundation of Tianjin, China (Nos. 16JCYBJC15500 and 17JCZDJC30500), and the Open Project Foundation of Information Security Evaluation Center of Civil Aviation, and Civil Aviation University of China (No. CAAC-ISECCA-201702).

| Web browser        | Length |
|--------------------|--------|
|                    | 10     | 20    | 50     | 100    | 200    | 500    | 1000   |
| Google Chrome      | 0.216  | 0.432 | 0.868  | 1.505  | 2.821  | 6.644  | 13.356 |
| 360 browser        | 0.223  | 0.427 | 0.849  | 1.519  | 2.792  | 6.703  | 14.172 |
| Baidu browser      | 0.219  | 0.424 | 0.853  | 1.532  | 2.835  | 6.693  | 13.349 |

Fig. 12: The time cost for SecureData to encrypt message.
References

[1] W. Alcorn, C. Frichot, and M. Orru, *Browser Hacker’s Handbook*. Indianapolis, IN, USA: John Wiley & Sons, 2014.

[2] J. Kiesel, B. Stein, and S. Lucks, A large-scale analysis of the mnemonic password advice, in *Network and Distributed System Security Symp.*, San Diego, CA, USA, 2017.

[3] D. L. Wheeler, Zxcvbn: Low-budget password strength estimation, in *Proc. 25th USENIX Security Symp.*, Austin, TX, USA, 2016, pp. 157–173.

[4] M. Dell, P. Michiardi, and Y. Roudier, Password strength: An empirical analysis, in *Proc. IEEE INFOCOM*, San Diego, CA, USA, 2010, pp. 1–9.

[5] C. S. Yuan, X. M. Sun, and R. Lv, Fingerprint liveness detection based on multi-scale LPQ and PCA, *China Commun.*, vol. 13, no. 7, pp. 60–65, 2016.

[6] T. H. Ma, J. J. Zhou, M. L. Tang, Y. Tian, A. Al-Dheilhaan, M. Al-Rodhaan, and S. Lee, Social network and tag sources based augmenting collaborative recommender system, *IEICE Trans. Inf. Syst.*, vol. E98-D, no. 4, pp. 902–910, 2015.

[7] S. D. Xie and Y. X. Wang, Construction of tree network with limited delivery latency in homogeneous wireless sensor networks, *Wirel. Pers. Commun.*, vol. 78, no. 1, pp. 231–246, 2014.

[8] W. He, D. Akhawe, S. Jain, E Shi, and D. Song, ShadowCrypt: Encrypted web applications for everyone, in *Proc. 2014 ACM SIGSAC Conf. Computer and Communications Security*, Scottsdale, AZ, USA, 2014, pp. 1028–1039.

[9] Y. J. Ren, J. Shen, J. Wang, J. Han, and S. Y. Lee, Mutual verifiable provable data auditing in public cloud storage, *J. Internet Technol.*, vol. 16, no. 2, pp. 317–323, 2015.

[10] J. Shen, H. W. Tan, J. Wang, J. W. Wang, and S. Y. Lee, A novel routing protocol providing good transmission reliability in underwater sensor networks, *J. Internet Technol.*, vol. 16, no. 1, pp. 171–178, 2015.

[11] Z. J. Fu, J. G. Shu, J. Wang, Y. L. Liu, and S. Y. Lee, Privacy-preserving smart similarity search based on Simhash over encrypted data in cloud computing, *J. Internet Technol.*, vol. 16, no. 3, pp. 453–460, 2015.

[12] Team of LastPass, Cross site scripting vulnerability reported, https://blog.lastpass.com/2011/02/cross-site-scripting-vulnerability-reported-fixed.html/, 2011.

[13] J. Lee, RobotForm everywhere, http://www.robobformcom/ everywhere, 2018.

[14] B. Stock and J. Martin, Protecting users against XSS-based password manager abuse, in *Proc. 9th ACM Symp. on Information Computer and Communications Security*, Kyoto, Japan, 2014, pp. 183–194.

[15] M. Zhao and C. Yue, Toward a secure and usable cloudbased password manager for web browsers, *Comput. Secur.*, vol. 46, pp. 32–47, 2014.

[16] Z. W. Li, W. R. He, D. Akhawe, and D. Song, The emperor's new password manager: Security analysis of Web-based password managers, in *Proc. 23rd USENIX Security Symp.* (USENIX Security 14), San Diego, CA, USA, 2014, pp. 465–479.

[17] Team SIK, Password-manager-apps, https://teamsik.org/trentportfolio/password-manager-apps/, 2017.

[18] Team of 1password, Go ahead. Forget your passwords, https://1password.com/, 2017.

[19] W. Melicher, B. Ur, S. M. Segreti, S. Komanduri, L. Bauer, N. Christin, and L. F. Cranor, Fast, lean, and accurate: Modeling password guessability using neural networks, in *Proc. 23rd USENIX Security Symp.*, Austin, TX, USA, 2016, pp. 175–191.

[20] B. Ur, S. M. Segreti, L. Bauer, N. Christin, L. F. Cranor, S. Komanduri, D. Kurilova, M. L. Mazurek, W. Melicher, and R. Shay Measuring real-world accuracies and biases in modeling password guessability, in *Proc. 24th USENIX Conf. Security Symp.*, Washington, DC, USA, 2015, pp. 463–481.

[21] D. Wang, Z. J. Zhang, P. Wang, J Yan, and X. Y. Huang, Targeted online password guessing: An underestimated threat, in *Proc. 2016 ACM SIGSAC Conf. Computer and Communications Security*, Vienna, Austria, 2016, pp. 1242–1254.

[22] S. Ruoti, D. Zappala, and K. Seamons, MessageGuard: Retrofitting the web with user-to-user encryption, http://arxiv.org/abs/1510.08943v1, 2015.

[23] Z. J. Fu, K. Ren, J. G. Shu, X. M. Sun, and F. X. Huang, Enabling personalized search over encrypted outsourced data with efficiency improvement, *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 9, pp. 2546–2559, 2016, doi: 101109/TPDS20152506573.

[24] Z. J. Fu, X. M. Sun, Q. Liu, L. Zhou, and J. G. Shu, Achieving efficient cloud search services: Multi-keyword ranked search over encrypted cloud data supporting parallel computing, *IEICE Trans. Commun.*, vol. E98-B, no. 1, pp. 190–200, 2015.

[25] Z. L. Liu, C. F. Jia, J. W. Li, and X. C. Cheng, Format-preserving encryption for datetime, in *Proc. 2010 IEEE Int. Conf. Intelligent Computing and Intelligent Systems*, Xiamen, China, 2010, pp. 201–205.

[26] B. Morris, P. Rogaway, and T. Stegers, How to encipher messages on a small domain, in *CRYPTO 2009*, S. Halevi, ed. Santa Barbara, CA, USA: Springer, 2009, pp. 286–302.

[27] B. Morris, P. Rogaway, and T. Stegers, How to encipher messages on a small domain, in *CRYPTO 2009*, S. Halevi, ed. Santa Barbara, CA, USA: Springer, 2009, pp. 286–302.

[28] Z. L. Liu, C. F. Jia, J. Yang, and K. Yuan, Format-preserving fuzzy query mechanism, in *Proc. 2013 4th Int. Conf. on Emerging Intelligent Data and Web Technologies*, Xi’an, China, 2013, pp. 220–226.

[29] W. He, D. Akhawe, S. Jain, E Shi, and D. Song, ShadowCrypt: Encrypted web applications for everyone, in *Proc. 27th USENIX Security Symp.*, San Diego, CA, USA: Springer, 2018, pp. 2706–2716, doi: 101109/TIFS20162596138.
[31] Z. H. Xia, X. H. Wang, L. G. Zhang, Z. Qin, X. M. Sun, and K. Ren, A privacy-preserving and copy-deterrence content-based image retrieval scheme in cloud computing, *IEEE Trans. Inf. Forensics Secur.*, vol. 11, no. 11, pp. 2594–2608, 2016.

[32] T. Spies, Feistel finite set encryption mode, https://csrc.nist.gov/csrc/media/projects/block-cipher-techniques/documents/bcm/proposed-modes/ffsem/ffsem-spec.pdf, 2008.

[33] Y. X. Dai and J. Steinberger, Indifferentiability of 8-round feistel networks, in *Proc. 36th Annu. Int. Cryptology Conf.*, Santa Barbara, CA, USA, 2016, pp. 95–120.

**Shuang Liang** received the bachelor degree from Nankai University, China, in 2015. Currently, she studies for the master degree in computer science at Nankai University. Her research interests include applied cryptography and data privacy protection.

**Yue Zhang** received the bachelor degree from Xiamen University, China, in 2016. Nowadays, he studies in computer science at Nankai University as a postgraduate. His research interests include database encryption system and searchable encryption.

**Zheli Liu** received the BS and MS degrees from Jilin University, China, in 2002 and 2005, respectively. He received the PhD degree in computer application from Jilin University in 2009. After a postdoctoral fellowship in Nankai University, he joined the College of Computer and Control Engineering of Nankai University in 2011. Currently, he works at Nankai University as an associate professor. His current research interests include applied cryptography and data privacy protection.

**Bo Li** obtained the bachelor degree from Nankai University in 2017. Currently, she studies for the master degree in computer science at Nankai University. Her research interests include applied cryptography and data privacy protection.

**Xiaojie Guo** received the bachelor degree from Nankai University in 2017. Currently, he studies for the master degree in computer science at Nankai University. His research interests include applied cryptography and data privacy protection.

**Chunfu Jia** got the PhD degree from Nankai University in 1996. He has finished the post doctoral research from the University of Science and Technology of China. Now he is a professor in Nankai University, China. His current interests include computer system security, network security, trusted computing, malicious code analysis, etc.