SGX-Cube: An SGX-Enhanced Single Sign-On System against Server-side Credential Leakage

Songsong Liu[1], Qiyang Song[2], Kun Sun[1], and Qi Li[2]

[1] George Mason University, Fairfax, USA
{sliu23,ksun3}@gmu.edu
[2] BNRist, Tsinghua University, Beijing, China
ashes.sqy126@gmail.com, qli01@tsinghua.edu.cn

Abstract. User authentication systems enforce the access control of critical resources over Internet services. The pair of username and password is still the most commonly used user authentication credential for online login systems. Since the credential database has consistently been a main target for attackers, it is critical to protect the security and privacy of credential databases on the servers. In this paper, we propose SGX-Cube, an SGX-enhanced secure Single Sign-On (SSO) login system, to prevent credential leakage directly from the server memory and via brute-force attacks against a stolen credential database. When leveraging Intel SGX to develop a scalable secure SSO system, we solve two main SGX challenges, namely, small secure memory size and the limited number of running threads, by developing a record-based database encrypted scheme and placing only authentication-related functions in the enclave, respectively. We implement an SGX-Cube prototype on a real SGX platform. The experimental results show that SGX-Cube can effectively protect the confidentiality of user credentials on the server side with a small performance overhead.

Keywords: SSO · SGX · Credential Leakage.

1 Introduction

User authentication is a common security mechanism in Internet applications to restrict unauthorized access to member-only areas on websites. Username/password is still the most commonly used user authentication method for online login systems. After being securely delivered to the authentication system, the user inputted username and password are validated by checking with the credentials stored in a credential file or database.

It continues as a challenge for Internet service providers to protect user credentials including passwords, which may be leaked out from either non-volatile storage (e.g., hard disk) or volatile storage (e.g., RAM). After breaking into the target server system, attackers can dump the credential database and then launch offline brute force attacks. Alternatively, attackers can manage to steal user credentials in plaintext from RAM. For instance, attackers can remotely read the memory content from victim servers by exploiting Heartbleed bug in the

OpenSSL library [6]. Even worse, when an attacker successfully breaks into the victim server, it can observe the entire credential verification process and easily retrieve credentials from memory. An advanced persistent attacker may collect most user credentials after stealthily residing in the server for a long enough time. In cloud environments, curious-but-honest service providers have the privilege to capture sensitive data in the memory of virtual machines, so it becomes another security concern on protecting user credentials in untrusted clouds.

Considering the difficulty in protecting user credentials, more Internet service providers choose to mitigate the management of various usernames and passwords by using Single Sign-On (SSO) services provided by third-party trusted companies such as Google [1] and Facebook [27]. By verifying a single credential on the SSO site, one user can obtain different authorized tokens to access multiple Internet services. It requires both users and Internet service providers to trust the third-party SSO sites with their credentials. However, similar to traditional online login systems, the SSO service providers are also troubled by the credential leakages from either hard disk or RAM [26]. Protecting the user credential in the SSO system still requires more effort.

Recently, researchers focus more on preventing information leakage during data processing. For instance, homomorphic encryption schemes [7] can ensure the credentials staying in ciphertext when being processed. Though it is promising to enhance the security of sensitive data in memory dramatically, it has to further reduce the overhead before being widely deployed. Another trend is to process sensitive data in an isolated and trusted execution environment. Thus, even if the host OS is malicious, the sensitive data can be processed in trusted environments securely. For instance, Intel Software Guard Extensions (SGX) provides a process-level isolation mechanism to protect user-level sensitive code and data from malicious OS [12]. On the client side, SGX has been used to protect password managers [8]. On the server side, SGX has been used to protect a credential encryption module [15]. However, it still requires further studies on using SGX to protect SSO services.

In this paper, we develop SGX-Cube, an SGX-enhanced secure SSO system, to protect user credentials on SSO servers. It can not only successfully prevent credential leakage from memory, but enhance the security of credential databases against offline brute-force attacks. Our system consists of three major components, namely, *authentication server*, *credential database*, and *application server*. As the core of SGX-Cube, the authentication server runs inside the SGX enclave to process authentication requests. It protects the credentials in the memory even if the host OS is compromised. When a user requests to access an application server, the login request will be forwarded to the authentication server. After successful authentication, the authentication server generates and delivers an authorization code to the user. Then the user uses this authorization code to request the corresponding token and access the desired service from the application server.

We implement a prototype of SGX-Cube on a computer supporting SGX v1 instruction set. To protect the transmission of credentials, we implement an HTTPS server inside the enclave. We use a lightweight database management

system SQLite as the credential database and a lightweight web server as the application server supporting OAuth 2.0 scheme. SGX-Cube is flexible to support other database systems and application servers. Our test-bed supports up to 4 threads in an enclave concurrently. The experimental results show that SGX-Cube introduces an average 0.6× extra time cost for a single thread in the authentication server. For a single request, it only takes about 1.5 *ms* for each authentication thread to complete all its tasks. For concurrent 500 requests, the average request processing time is about 1.7 *ms*. Our security analysis shows that SGX-Cube can effectively increase the security of the SSO system by preventing credential leakage from both memory and hard disk. In summary, we make the following contributions:

- We propose SGX-Cube, an SGX-enhanced secure SSO system, to increase the security and privacy of user credentials on the server by placing operations on credentials inside the SGX enclave. We further propose a record-based encryption scheme to improve authentication efficiency.
- We formulate the security of SGX-Cube in two aspects: confidentiality and integrity. Then, we analyze the security of SGX-Cube against both online attacks and offline attacks.
- We implement a prototype of SGX-Cube using SGX v1. The experimental results show that it is a practical solution with a small performance overhead.

2 Background

2.1 Intel SGX

Intel Software Guard Extensions (SGX) [12] provides user-level isolated execution environments (enclave) to protect the confidentiality and integrity of application code and data in a reserved memory region named the Enclave Page Cache (EPC), which is encrypted and authenticated by a Memory Encryption Engine (MEE) hardware module. SGX protects an application against illegal access from other applications, OS, and hypervisor. An SGX application is divided into two components: *a trusted component* and *an untrusted component*. The trusted component contains the code and data that need to be protected inside the enclave, while the untrusted component contains the rest part. To bridge these two components, SGX uses the enclave entry call (Ecall) and outside call (Ocall) mechanisms, where Ecall is the function call that enters the enclave from outside and Ocall is the function call that calls an untrusted outside function from an enclave. The code inside an enclave can only be executed in the user mode. The maximum EPC size is limited (128 MB for SGX v1, 256 MB for SGX v2). When the configured enclave size is larger than the EPC size, the performance overhead becomes inevitably high due to paging between EPC and normal memory.

2.2 Single Sign-on (SSO) Systems

A single sign-on (SSO) system provides an authentication process that allows a user to access multiple application servers with one set of login credentials. Therefore, a user only needs to log in once and then gain access to different applications without re-entering the login credentials at each application server.

As third-party authentication systems, the SSO systems are trusted by both end users and a number of application servers. One SSO system contains three main parties: *user*, *identity provider (IDP)*, and *relying party (RP)*, where RPs are the applications/websites to be accessed by the users and the IDPs are responsible for providing the authentication services. The workflow is described as follows. First, the client connects to the RP (i.e., application server), which then sends the authorization request to the client. Next, after verifying the IDP (i.e., authentication server) via remote attestation, the client sends its login name and password to the IDP. After successfully verifying the client, the IDP generates an authentication token and sent it to the client. Next, the client forwards the authentication token to the RP. After verifying the client with the provided authentication token, the RP can request the user information (e.g., username) from the IDP and grant the services to the client.

3 Threat Model and Assumption

We focus on protecting the server-side login process and ensuring the confidentiality of user credentials on the server side. In this work, we refer user credentials to the username and password only, though other credentials may also include sensitive information such as PIN or credit card information. The credentials experience three states in the complete login procedure, namely, data-in-motion, data-in-use, and data-in-rest. Therefore, the attacker may commit a series of attacks against each state of credentials to defeat the user authentication process and collect valuable username/password and other user credential information.

In our threat model, we consider a strong attacker, who can commit not only the offline attack but also the online attack. To commit the offline attack, the attacker can extract the credential database from the server, then analyze it with the known information to infer other sensitive information or even brute force the encrypted credentials directly. To commit the online attack, the attacker would compromise the authentication server and users. It targets both the data-in-motion and the data-in-use. The attacker may retrieve plaintext credentials by eavesdropping the communication channels connected to the authentication server, monitoring the memory of the server, manipulating login requests, or creating new known-plaintext records in our database to launch the chosen-plaintext attack. If it is result-less to reveal the desired credentials, the attacker may try to circumvent the authentication process via manipulating the authentication process in the memory or splicing stored credential records.

We assume the Intel SGX can be trusted. Although the SGX has become vulnerable to high-cost side-channel attacks [18], lots of efforts have been made to mitigate these attacks in both software [22] and hardware [13, 19]. We target at protecting the user credentials on the server side, and the credential on the client side can be protected by other SGX-based solution [8].

4 System Design

4.1 System Overview

Figure 1 shows the overall architecture of our SGX-enhanced secure SSO login system. When an application server attempts to use our SSO service, its first

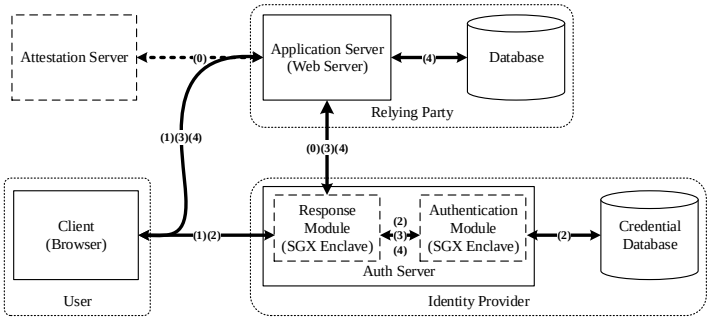


Fig. 1. The Architecture of SGX-Cube. (0) Attestation; (1) Login Request; (2) Credential Handling; (3) Authorization Grant; (4) Service Access.

step is to authenticate the SSO authentication server. It will request a remote attestation to ensure the integrity of the authentication server running in the SGX enclave. An attestation server may be required to facilitate the attestation [12]. After a successful remote attestation, it can verify the authentication server.

Based on the standard SSO scheme, our SGX-Cube is divided into three main components: Identity Provider (IDP), Relying Party (RP), and user. These three components interact with each other during the complete login procedure.

Identity Provider (IDP). As the core component of the SSO login system, the IDP consists of two main components, namely, an *enclave-based authentication server* and a *credential database*.

The authentication server runs inside the enclaves. It is responsible for conducting remote attestation, receiving the user's login credentials from clients, verifying user login credentials against the credential database, generating an authentication token, and then sending the token to both the client to facilitate the authentication of the client to the application server. Here, the user login credential contains the username and the password. The authentication token can be customized according to the requirements of the specific application server. The authentication server also handles new user registration, password update/reset, and account revocation. Depending on its function, we split the authentication server into two modules in two separate enclaves: the response module and the authentication module. Splitting the authentication server into two enclaves, we intend to mitigate risks of the credential processing (authentication module) by isolating it from the potential vulnerabilities of network interaction (response module). The response module is to establish secure communication channels with the outside (the application server and the client) to defend eavesdropping and tampering. It handles all the requests from them. The authentication module processes credential related tasks. During the login procedure, all the credentials are passed from the response module to the authentication module via a secure communication channel of intra-platform.

The credential database stores user credential information for user authentication. The privacy of the database is protected when stored on the hard disk. Instead of encrypting the entire database with one key, we protect each column of the database table with a unique key (subkey). The subkeys are encrypted

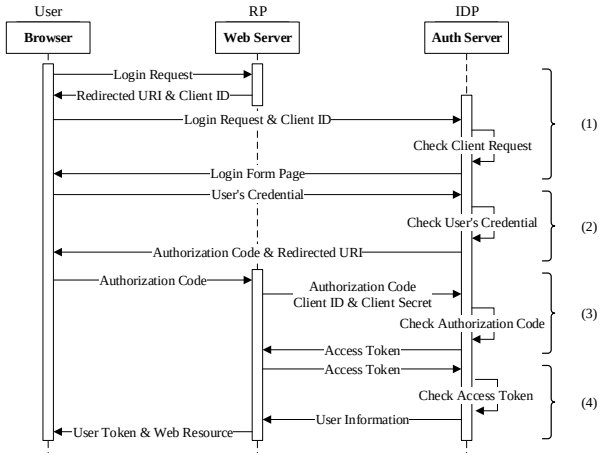


Fig. 2. The Flow of user Login Process

when stored on disk and in normal memory and are only decrypted inside the enclave. Towards achieving a record-based encryption, the fields of one record are bound together [5]; see Section 4.3. By using record-based encryption, the query operations can be applied to the encrypted values directly. The credential database can be organized by the default DBMS (database management system) without any modification. After calculating the encrypted value of the inputted username, the authentication server queries the database and reads the encrypted value of the corresponding password into the enclave.

Relying Party (RP). The RP could be any kind of off-the-shelf application server on the internet. It provides Internet services to the user after the user is correctly authenticated by the IDP. All the login requests are redirected to the IDP conducting the credential verification. It only receives the authorization code from the user and verifies the code from IDP via a secure communication channel. If the code is correct, the IDP would allocate the token and corresponding user information to the application server. The user information is used as the user identity on the application server. Based on this identity in its database, the application server creates or updates the record for the authorized user.

User. In our design, the user should get a seamless, out-of-the-box, easy to use client. The client could be a typical browser or self-developed application that supports SSO scheme. It assists user to get the authorization from IDP and access the desired Internet service of RP.

4.2 Login Procedure

In this section, we discuss each stage of the login procedure. We divide a complete login procedure into four stages, as shown in Figure 2. While the remote attestation stage is not a part of the login procedure, it is an essential job to be conducted before the login procedure.

Stage 0: Attestation. Before the application server accepts a login request, it is necessary to ensure that the IDP is trustworthy. It conducts the remote attestation to verify that 1) the enclave is running on a genuine SGX enabled

platform, and 2) the authentication server code running inside the enclave has not tampered with [14]. Literally, the application server should attest to both two modules of the authentication server. However, since the authentication module is not accessible from the outside, the remote attestation is only conducted on the response module directly. To solve this problem, we let the response module and authentication module to conduct the local attestation [28] for each other during the initialization. Until the dual local attestation is successful, the response module will accept the remote attestation request. The response module will generate a response that contains a signed hash value of the software running inside the enclave and the enclave environment. According to Intel [23], the application server needs to forward the remote attestation response to the response module, which stores the keying material to verify the hash value. To ensure the trustworthiness of IDP, the application server could start the remote attestation periodically (e.g., 24 hours). Otherwise, the remote attestation could be conducted by a trusted third-party CA. The application server only needs to verify the certification of the authentication server.

Stage 1: Login Request. When a user requests to access the resource of the application server, the application will check if the request contains a valid token generated by itself. If yes, the user can access the desired Internet service directly. Otherwise, the login request is forwarded to the authentication server. A secure communication channel is established between the user client and the authentication server. The certification of the authentication server will be checked during the connection construction phase. Then the user client sends the login request to the authentication server via the new-built secure communication channel. The authentication server replies to the login request with a login interface (i.e., web page), which will be shown on the user's client.

Stage 2: Credential Handling. After the secure communication channel is established, the user can input the credential on the client side. The secure channel ensures that the user credential is directly delivered into the enclave on the authentication server. Once the authentication server receives the user credential, it encrypts the username and uses it as the key to query the corresponding record from the credential database. The encrypted password is read into the enclave to conduct the password verification. The plaintext of the password won't leak out of the enclave. If the credential is valid, the authentication server will generate an authorization code and return it to this user.

Stage 3: Authorization Grant. For the user, this stage is transparent. No further operations are required. After receiving the authorization code from the authentication server, the user client will resend it to the application server automatically. The application server uses this authorization code to request the access token from the authentication server. Similar to stage 2, a secure communication channel is established between the application server and the authentication server. The authentication server verifies the authorization code. If valid, an access token is generated and allocated to the application server.

Stage 4: Service Access. In this stage, the application server needs to request user information from IDP. The authentication server receives the request of

user information with the access token from the application server. If the access token is valid, the application server will receive corresponding user information. The user information is used as the user identity on the application server. The application server stores the user information in its own database allocates a user token and opens the resource access entrance to the user. After that, the user can access the desired Internet service. The whole login procedure is completed.

4.3 Credential Storage

The stored user credentials include three major elements: username, password, and user information. In our scheme, both username and password are encrypted, which increases the entropy of user credentials against brute-force attacks. Moreover, the username may contain sensitive information and provide the attackers with a good hint, directly or indirectly, to speed up the password cracking process [16]. The user information stored in the credential database could be username or other information (e.g., e-mail and address) used in the application server. It is an identity of the user for the application servers.

Since the database is saved on the hard disk, the user credential data inside the database are protected by encryption mechanisms. We use three different subkeys K_u, K_p, and K_i to protect the three columns, i.e., username, password, and user information, respectively. All three subkeys are encrypted by the enclave seal key when stored on the hard disk. First, in the username column, we store the HMAC of the username with subkey K_u. Its saving value is

$$hmac(K_u, username) \tag{1}$$

Since the username serves as the primary key of the database, it should be unique. Next, for the password column, we compute the hash of the combination of subkey K_p and username as the key for the HMAC of password. The password column saves the following value

$$hmac(hash(K_p, username), password) \tag{2}$$

Finally, for the user information column, we encrypt this field with a symmetric encryption scheme. Its key is the hash value of the combination of subkey K_i and username. The saved value is

$$\{userinfo\}_{hash(K_i, username)} \tag{3}$$

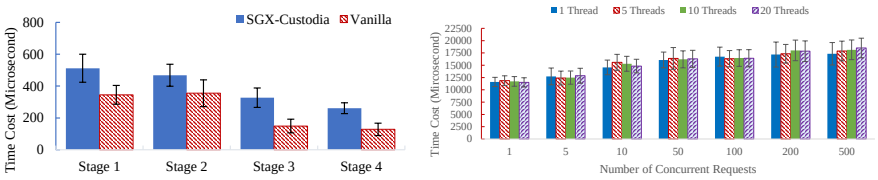
We bind the username into both the password and user information to eliminate the possibility of field substitution. It is also equivalent to protect each record with a different key.

5 System Implementation

We develop the SGX-Cube prototype on SGX SDK v2.1.3 [12]. We implement our authentication server running inside the enclave by using around 4K LOC. The total size of binary loading into enclave memory is about 9.8 MB, which includes an OpenSSL library [9]. We use the HTTPS on all communication channels to protect the data transmission and ensure that credentials enter the enclave of the authentication server. We use a lightweight relational database management system, SQLite v3.13.0, as the credential database. We develop a lightweight web server based on python Flask framework v0.12.2 as the application server.

Our implementation follows standard Authorization Code Grant type of OAuth 2.0 [10]. We use 256-bit SHA256 HMAC to protect username and password and 128-bit AES in counter mode to protect user information In our prototype, the authentication server and the credential database are deployed on the same machine, while the application server and clients are deployed on another machine in the same local area network.

6 Performance Evaluation



(a) Authentication Server Response (b) Complete Login Procedure

Fig. 3. Time Cost of SGX-Cube

We evaluate the authentication server performance and overall performance of our SGX-Cube prototype, respectively. We deploy the authentication server on a machine with Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz and 8GB RAM. Its size of reserved EPC is set to 128 MB (about 93.5 MB available for applications) on BIOS. The web server and user browser are deployed on another machine with Intel(R) Core(TM) i7-4790 CPU @ 3.6 GHz and 16GB RAM. All these two machines run 64-bit Ubuntu Linux 16.04 with kernel 4.15.

6.1 Authentication Server Performance

The authentication server replies to different types of requests in each stage. Therefore, we measured the performance of the authentication server in the login stage. Since there is no built-in timer function in SGX v1, we use the timer function outside the enclave via Ocall. The Ocall introduces the extra overhead around $15\mu s$. The extra overhead is compensated in the final results. We run each experiment 100 times and calculate the average values.

Figure 3(a) shows the measurement of the authentication server in each stage. From it, we can observe that Stage 1 has the highest time cost. The authentication server sends a complete HTML login page to the user. Although we only implemented a simplified login page, it is still the largest size of data to be sent in all four stages. As mentioned in Section 4.2, the authentication server needs to handle the received credential and access the database in Stage 2. The database access causes the main overhead in this stage. In Stage 3 and 4, the authentication server locates the required information and responses to the corresponding request. Since the authorized user information has been extracted from the credential database and cached inside the enclave, the time cost of Stage is relatively low. We re-implement an authentication server without using SGX as the vanilla SSO system. Compare with it, the overhead of SGX-Cube increases about 0.5× (Stage 1), 0.3× (Stage 2), 1.1× (Stage 3) and 1× (Stage 4). The total overhead increases by about 0.6×.

6.2 Overall Performance

To measure the overall time consumption of a complete login procedure, we record the time length from the user starts the login request to user gains the access of web server. The overall overhead includes the authentication server response, the webserver request handling, and user browser redirection. We measure the time cost of a complete login procedure with a various number of concurrent requests. We also evaluate the performance with multiple threads. The response module and authentication module keep the same number of threads. Note that the number of authentication server threads is limited by the size of the enclave in SGX v1, which includes heap, stack, code/text segments, etc. Although the SDK source code sets 128GB as the maximum enclave size for the 64-bit program, the enclave size cannot reach this theoretical maximum since the driver did not support the Version Array (VA) page swapping yet. In our experiment, we set 1MB max heap size and 256KB max stack size for the enclave. The timer is on the user client. Each thread in the experiment handles 100 requests in total.

As shown in Figure 3(b), a complete login procedure costs about $11500\mu s$ for a single authentication server thread and a single request. The percentage of authentication server overhead is about 14%. As the number of concurrent requests increases, the average time costs for each request increase. That is because the authentication server cannot handle arriving requests in time. The requests have to wait in the queue. Although we could add more threads in the authentication server, the performance is not improved significantly. Even multiple threads can be created in the authentication server, the number of threads, that can process the login requests concurrently, is limited and decided by the platform. When the number of threads exceeds the number of logical processors, the extra threads will sit idle. In this case, the concurrent requests are processed in sequence to some degree. We will discuss how SGX-Cube could deal with this limitation of concurrent requests in system scalability (Section 8).

7 Security Analysis

In this section, we give security definitions and briefly demonstrate the security of SGX-Cube under two types of attacks: offline and online attacks. More details can be found in [25].

7.1 Data Confidentiality under Offline Attacks

As the records of credential databases are encrypted under secret keys, attackers cannot derive any sensitive information from encrypted records without knowing the keys. To demonstrate the data confidentiality of entire databases under offline attacks, we adopt real world versus ideal world formalization [4] to define the column confidentiality under offline attacks. It is parameterized by a stateful leakage function $\mathcal{L}_1$ describing what information leaks in the protocols. More precisely, we define two games $Real_{\mathcal{A}}$ and $Ideal_{\mathcal{A}}$ with a simulator $\mathcal{S}$ [17] and an adversary $\mathcal{A}$. The simulator $\mathcal{S}$ can simulate real protocols and data using a leakage collection, and the adversary $\mathcal{A}$ has the server's view and can interact with real (or simulated) protocols. If $\mathcal{A}$ cannot distinguish the simulated column data from the real column data, then we can say the column achieves $\mathcal{L}_1$-confidentiality

under offline attacks. Note that the columns of usernames, passwords, and user information are encrypted by the cryptographic tools HMAC and AES. Therefore, if the adversary $\mathcal{A}$ has finite computational resources and has not held the secret subkeys of each column, it cannot launch chosen plaintext attacks to distinguish the simulated column data from real column data.

7.2 Data Confidentiality under Online Attacks

Similar to the column confidentiality under offline attacks, the column confidentiality under online attacks is also captured by real world versus ideal world formalization with an attacker $\mathcal{A}_2$ and a simulator $\mathcal{S}$. The attacker $\mathcal{A}_2$ is online, and it has a stronger capability than the online attacker. Specifically, it can compromise both the server and a subset of users and can utilize them to launch chosen-plaintext attacks. Since $\mathcal{A}_2$ can control users to launch chosen-plaintext attacks, it can input arbitrary plaintexts into users' programs and then observe output ciphertexts. Note that the column data of usernames are encrypted by the same subkey, and the subkey is held by all users. Therefore, $\mathcal{A}_2$ can distinguish the simulated column data of usernames from the real column data by running a user's programs. As the column data of passwords and user information are encrypted under the subkeys of different users, if $\mathcal{A}_2$ do not know corresponding subkeys, it cannot distinguish the simulated column data of passwords and user information from the real data.

7.3 Data Integrity under Online Attacks

Since data can only be manipulated by online attacks, we only demonstrate data integrity under online attacks.

Data Integrity in Memory. The data integrity in memory is guaranteed by the security of the SGX enclave. The whole authentication procedure is completed inside the enclave. The authentication results are delivered to the requester from the enclave directly via secure communication channels. The results are not revealed in the server memory and cannot be tampered. Hence, the attacker can't compromise the data integrity in the memory.

Data Integrity on Disk. The data integrity on disk means that credential databases cannot be manipulated to authenticate a user without a correct pair of username and password. Here, we consider an attacker $\mathcal{A}_3$ who can corrupt both authentication server and a subset of users. Particularly, $\mathcal{A}_3$ can control a user u_1 to generate a password p_1 from a known string, and then replace an honest user u_2's password with p_1 in the credential database. Next, $\mathcal{A}_3$ may attempt to impersonate u_2 by sending the known string as the password. Recall that the HMAC value of each user's password is generated by a unique secret key. Therefore, attackers have a non-negligible advantage to impersonate an honest user if *HAMC_SHA256* is collision-resistant.

8 Practical Usage

Enclave Migration. In traditional enterprise networks, we can deploy a dedicated physical machine as the authentication server, which is not moved frequently. However, when the entire network is in the cloud environment, authentication server migration should be supported. Usually, the administrators only need to

perform an offline server migration by shutting down an enclave on one physical machine and rebooting it on another physical machine, which has different embedded SGX keying materials. Our system design enables a smooth offline enclave migration. First, The credential database can be directly copied or linked to the new enclave without any changes. Second, the subkeys can be securely sent to the new enclave via either a direct secure network connection or an out-of-band communication channel. The new enclave encrypts the subkeys with its own seal key and then saves them on its local storage. We can also support enclave live migration by adopting the migration mechanism proposed by Alder et al. [2].

System Portability. The authentication server can be implemented on any SGX-enabled Intel platform. Moreover, due to the modular design and clear interfaces between modules, our system is flexible to support various Internet services and different database systems. First, in addition to web servers in our prototype system, the application servers could be other types of Internet service, such as mail servers, FTP servers, cloud storage servers, and so on. All they need is a suitable interface supporting the standard SSO scheme. The application server only needs to support a secure communication channel with the authentication server and the client and handle the authentication token, respectively. Second, our system design is flexible to integrate various database systems to store user credentials. Our prototype system uses a lightweight database SQLite, which is good at storing data locally. To achieve a better data management capability, more powerful database systems can be adopted, such as MySQL, SQL Server.

System Scalability. The number of threads in one enclave is limited. Our test-bed allows configuring at most 7309 threads, given the 1 MB maximal heap size and 256 KB maximal stack size. In practice, the number of active threads is much smaller than that number. One straightforward solution is to increase the number of active threads by deploying multiple enclaves on the same platform, while those enclaves still share the same EPC. The number of threads running simultaneously is decided by the number of logical CPU cores. Hence, when a large number of requests arrive around the same time, only the first several requests are served, and others have to wait. To serve tens or hundreds of login requests concurrently, we may deploy multiple SGX cards [3] or physical servers.

9 Related Work

To prevent credential leakage, the login credentials must be protected securely on the client, the server, and the transmission channel. On the client side, the credential security mainly depends on how well users could protect their end devices. SGX has been used to protect password managers, which stores all the passwords of end-users on the client [8]. To protect the transmission channel, mature secure protocols (e.g. TLS/SSL) are adopted. On the server side, the key issue is how to process and store the user credentials securely. By porting the credential processing code in trusted execution environments such as SGX, attackers cannot manipulate the control flow of authentication. To protect credential storage, the common solution is database encryption [24]. Particularly, subkeys have been used to protect different data records or tables in the database [5, 11]. CryptDB [21] and Seabed [20] are the database systems

supporting encrypted query. Homomorphic encryption [7] allows more other operations on the encrypted data, while the high cost still hinders its wide deployment. Using trusted hardware to protect the credentials is a viable solution with much smaller system overhead. SafeKeeper [15] is the first SGX-based solution for credential protection on the server side. It uses the encryption enclave to replace the PHPass MD5 hash function. To establish a secure channel, SafeKeeper uses DHKE to establish a shared encryption key between browser add-on and enclave. Compared to SafeKeeper, our solution provides a more flexible framework that can integrate with various internet services and database systems. The user can use the original browser without extra add-on installation.

10 Conclusion

This paper demonstrates SGX-Cube, an SGX-enhanced SSO system that targets at preventing user credential leaking from both memory and hard disk on the server side. By utilizing SGX as an isolated execution environment, we can protect the confidentiality of user credentials when they are processed in the memory. Besides, we can protect the control flow of the authentication process. We choose to protect both the username and password to further defeat offline brute-force attacks. We propose a simple but effective record-based encryption scheme to protect user credentials stored on the hard disk. Due to the modular design, it is flexible to port SGX-Cube onto various application servers and database systems. We implement a prototype of SGX-Cube on a real SGX platform. Our experiments show that SGX-Cube can effectively protect the confidentiality of login credentials with a small performance overhead.

Acknowledgments

This work is partially supported by U.S. ONR grants N00014-18-2893, N00014-16-1-3214, and N00014-20-1-2407.

References

1. Google Single Sign-On (accessed in Dec 2019), `https://cloud.google.com/identity/sso/`
2. Alder, F., Kurnikov, A., Paverd, A., Asokan, N.: Migrating sgx enclaves with persistent state. In: 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). pp. 195–206. IEEE (2018)
3. Chakrabarti, S., Hoekstra, M., Kuvaiskii, D., Vij, M.: Scaling Intel software guard extensions applications with Intel SGX card. In: Proceedings of the 8th International Workshop on Hardware and Architectural Support for Security and Privacy (2019)
4. Curtmola, R., Garay, J., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. Journal of Computer Security **19**(5), 895–934 (2011)
5. Davida, G.I., Wells, D.L., Kam, J.B.: A database encryption system with subkeys. ACM Transactions on Database Systems (TODS) **6**(2), 312–328 (1981)
6. Durumeric, Z., Li, F., Kasten, J., Amann, J., et al.: The matter of heartbleed. In: Proceedings of the 2014 conference on internet measurement conference. pp. 475–488. ACM (2014)
7. Gentry, C., Boneh, D.: A fully homomorphic encryption scheme, vol. 20. Stanford University Stanford (2009)

8. Goldberg, J.: Using Intel's SGX to Keep Secrets even Safer (2017), https://blog.1password.com/using-intels-sgx-to-keep-secrets-even-safer/
9. Han, J.: SGX-OpenSSL: Openssl library for SGX application (2017), https://github.com/sparkly9399/SGX-OpenSSL
10. Hardt, D.: Rfc 6749: The oauth 2.0 authorization framework. Internet Engineering Task Force (IETF) **10** (2012)
11. Hwang, M.S., Yang, W.P., et al.: Multilevel secure database encryption with subkeys. Data & knowledge engineering **22**(2), 117–131 (1997)
12. Intel: Intel Software Guard Extensions for Linux OS SDK (2018), https://github.com/intel/linux-sgx
13. Intel: Side Channel Mitigation by Product CPU Model (2018), https://www.intel.com/content/www/us/en/architecture-and-technology/engineering-new-protections-into-hardware.html
14. Johnson, S., Scarlata, V., Rozas, C., Brickell, E., Mckeen, F.: Intel software guard extensions: EPID provisioning and attestation services. White Paper **1**, 1–10 (2016)
15. Krawiecka, K., Kurnikov, A., Paverd, A., Mannan, M., Asokan, N.: Safekeeper: Protecting web passwords using trusted execution environments. In: Proceedings of the 2018 World Wide Web Conference on World Wide Web. pp. 349–358 (2018)
16. Li, Y., Wang, H., Sun, K.: A study of personal information in human-chosen passwords and its security implications. In: IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications (April 2016)
17. Lindell, Y.: How to simulate it–a tutorial on the simulation proof technique. In: Tutorials on the Foundations of Cryptography, pp. 277–346. Springer (2017)
18. Nilsson, A., Bideh, P.N., et al.: A survey of published attacks on Intel SGX (2020)
19. Orenbach, M., Baumann, A., Silberstein, M.: Autarky: closing controlled channels with self-paging enclaves. In: Proceedings of the Fifteenth European Conference on Computer Systems. pp. 1–16 (2020)
20. Papadimitriou, A., Bhagwan, R., Chandran, N., et al.: Big data analytics over encrypted datasets with seabed. In: OSDI. pp. 587–602 (2016)
21. Popa, R.A., Redfield, C., Zeldovich, N., Balakrishnan, H.: Cryptdb: protecting confidentiality with encrypted query processing. In: Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles. pp. 85–100. ACM (2011)
22. Sasy, S., Gorbunov, S., Fletcher, C.W.: Zerotrace: Oblivious memory primitives from intel sgx. IACR Cryptology ePrint Archive **2017**, 549 (2017)
23. Scarlata, V., Johnson, S., Beaney, J., et al.: Supporting third party attestation for Intel SGX with Intel data center attestation primitives. White Paper (2018)
24. Shmueli, E., Vaisenberg, R., Elovici, Y., Glezer, C.: Database encryption: an overview of contemporary challenges and design considerations. ACM SIGMOD Record **38**(3), 29–34 (2010)
25. Song, Q.: Sgx cube security analysis (2020), https://github.com/ashessqy126/SGX-Cube-Security-Analysis/blob/master/SGX-Cube-Security-Analysis.pdf
26. Winder, D.: Unsecured facebook databases leak data of 419 million users. WIRED (Sep 2019), https://www.forbes.com/sites/daveywinder/2019/09/05/facebook-security-snafu-exposes-419-million-user-phone-numbers/#3b32d5a1ab7f
27. Workplace by Facebook: Facebook Single Sign-On (accessed in Dec 2019), https://www.facebook.com/workplace/resources/tech/authentication/sso
28. Xing, B.C., Shanahan, M., Leslie-Hurd, R.: Intel software guard extensions (Intel SGX) software support for dynamic memory allocation inside an enclave. In: Proceedings of the Hardware and Architectural Support for Security and Privacy 2016, pp. 1–9 (2016)