Received February 16, 2021, accepted February 24, 2021, date of publication March 4, 2021, date of current version March 12, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3063770

SGX-UAM: A Secure Unified Access Management Scheme With One Time Passwords via Intel SGX

LIANGSHUN WU[1,2], **H. J. CAI**[1,2], **AND HAN LI**[2]
[1]School of Computer Science, Wuhan University, Wuhan 430079, China
[2]Zall Research Institute of Smart Commerce, Wuhan 430000, China

Corresponding author: H. J. Cai (hjcai@whu.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61832014.

ABSTRACT With the convergence of fixed and mobile networks, heterogeneous networks are becoming ubiquitous. Internet giants are seeing the plight of identity authentication. To address this issue, unified access management (UAM) was conceived. This paper provides a novel unified access management scheme, named SGX-UAM, with one-time passwords (OTPs) based on Intel software guard extensions (SGX). SGX-UAM outperforms generic UAM for providing resistance to most client attacks, man-in-the-middle (MITM) attacks, phishing attacks, most replay attacks and most denial of service (DoS) attacks to which generic UAM implementaions are vulnerable. Specifically, client attacks are prevented by ensuring input security and memory security, where the former is achieved through shuffle mapping and ''periodic hooking'' strategy, the latter is mainly guaranteed by Intel SGX; MITM attacks are prevented by transferring ciphertext rather than plaintext; phishing attacks are avoided by authorization control; replay attacks cannot succeed because we adopts OTPs, which contain time-related dynamic factors that expire in a few seconds; as for DoS attack, we blunted its edge by blocking-invocation for identical user connection. SGX-UAM also differs from generic UAM in that it relieves the security concerns of sevice providers (SPs) and protects users' privacy at little cost of performance. An exceptional value of SGX-UAM is that it brings a lightweight OTP solution that eliminates the need of additional hardware devices, thus reducing the costs. The experimental results show that SGX-UAM consumes almost the same time with OpenID and OAuth2.0 for one login request and performs steadily when handling sequential login requests. Furthermore, the resource usage for SGX-UAM is acceptable.

INDEX TERMS One-time password, Intel SGX, unified access management, security.

I. INTRODUCTION

Traditional identity and access management tools work well for addressing specific portions of an enterprise (specific app environments, as in on-premises or in the cloud; or specific users, as in employees vs. external partners) on their own. However, with the convergence of fixed and mobile networks, the coexistence of heterogeneous networks and the diversification of services have become prominent features of the Internet. At the beginning stage of convergence, networks and services exhibit strong independence and autonomy. When users access different networks and services, they need to be authenticated and authorized repeatedly, and this not only requires users to maintain multiple identity information sets but also adds some difficulty for operators in terms of managing system and charging users.

Unified access management (UAM) is an evolution of identity and access management (IAM) systems that provides unified login and identity information sharing services for different networks and business systems [1], and the idea originates from single sign-on (SSO) [2]. In 2010, the OAuth protocol was proposed, and this is a concrete implementation of UAM. The OAuth2.0 protocol was quickly adopted by many Internet platforms due to its security, convenience and other hallmarks [3]. However, UAM is deemed no longer secure now. In recent years, several security incidents have hitted UAM, for example, in March 2012, it is reported that 8 serious logic loopholes were found in high-profile identitiy providers (IDPs) and relying party websites, such as OpenID (including Google ID and PayPal Access), Facebook, Janrain, Freelancer, FarmVille, and Sears.com [4];

The associate editor coordinating the review of this manuscript and approving it for publication was Chakchai So-In.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

in May 2014, a vulnerability named "Covert Redirect" related to OAuth2.0 and OpenID was disclosed. And beyond inherent flaws, client attacks, man-in-the-middle (hereinafter called "MITM") attacks, phishing attacks, replay attacks, denial of service (hereinafter, "DoS") attacks, and so forth are undermining the usability of UAM.

Out of an fear of privacy breaches stemming from the insecurity of UAM, some influential service providers (SPs), who have already owned robust identity databases, are reluctant to entrust all authentication businesses to third-party operators. How to relieve this concern of SPs is a question that remains unanswered. As far as the users are concerned, privacy issues worry them more, because most UAM frameworks even do not give users any choices about releasing their personal information. To summarize, security and privacy concerns are putting SPs and users off UAM and changes must be made soon.

As an important component of single-factor, two-factor and three-factor authentication, one-time password (OTP) is considered an indispensable technology of UAM. Many UAM implementations, such as OAuth2.0, have already offered interfaces to enable OTP verification. Using OTPs, users do not need to remember passwords in a tedious manner; all they need to do is to hold a small device of uniform size and shape with a USB flash disk to be able to identify themselves for the service provider. The timeliness that an OTP need is serviceable in resisting the replay attacks suffered by UAM. Nevertheless, OTP technology is flawed, it requires the user to maintain a synchronizer or event counter, often via special hardware, which greatly increases the user's cost. Trusted Execution Environment (TEE) technologies, such as Intel SGX, might bring about a change for OTPs. SGX helps eliminating the need for additional hardware devices if we generate the OTP key with CPU-bound instruction (viz. EGETKEY), thus strictly guaranteeing the uniqueness of the device.

This paper strives to design a unified access management system based on Intel SGX, named "SGX-UAM", combining the merits of OTPs and UAM. Generally, UAM requires an SP and an IDP to establish authorization agreements. SGX-UAM suggests a novel paradigm for UAM bacause it assigns the burden of identity authentication to the users rather than the SPs. The decision is based upon two considerations:

1) Influential SPs, who own large identity databases, are not inclined to trust an IDP because it incurs disproportionate share of the risks relative to the limited convenience it brings; what's more, their authentication mechanisms are mature enough to be fully self-sufficient.
2) The users, though limited by resources, can act as the main undertakers of authentication tasks if the performance of SGX-UAM is acceptable. Moreover, the login credentials can be stored in ciphertext form in IDP that user privacy can be protected.

It can be said that the decision is a trade-off between commercial concerns, privacy-preservation and performance, and we suggest it is appropriate.

The main contributions of this paper include the following:

1) It proposes a highly secure unified access management scheme that resists most client attacks, MITM attacks, phishing attacks, most replay attacks and most DoS attacks to which generic UAM implementations are vulnerable. We have proven this theoretically and experimentally.
2) It assigns the burden of identity authentication to the users rather than the SPs, which not only relieves the security concerns of SPs but also protects users' privacy at little expense of performance.
3) It proposes a lightweight OTP scheme with no additional hardware required by resorting to Intel SGX technology, which reduces the cost of OTP.

The rest of the paper proceeds as follows: Section II outlines the requisite background knowledge; Section III reviews the related work; Section IV highlights the threat model and security claims; Section V presents our scheme; Section VI demonstrates theoretical security analysis; Section VII shows the experimental results. Finally, Section VIII draws the conclusion.

II. PRELIMINAY KNOWLEDGE

A. OTP

An OTP (one-time password) is a very convenient technical means for enhancing static password authentication, and it is an important part of two-factor authentication technology. Common OTP implementation schemes include HOTP (HMAC-based one-time password) and TOTP (time-based one-time password), which correspond to two RFC protocols, viz. RFC4266 [5] and RFC6238 [6], respectively.

Generally, the OTP calculation formula is

$$\sigma = \tau(\prod(\varepsilon, c)), \tag{1}$$

where ε represents key, c is a parameter; $\prod$ denotes using SHA-1 for HMAC, and τ is a function that intercepts the encrypted string.

HOTP sets c as an 8-byte common counter that requires synchronization between the authenticator and the authenticatee, and this is called a moving factor in RFC4266. In addition, the hash function of HOTP is expected to be SHA2, i.e., SHA-256 or SHA-512.

For TOTP, $c = (T - T_0)/X$, where T denotes the current Unix timestamp, T_0 is usually 0, and X represents how long it takes to generate an OTP, that is, the time interval (X is almost always set to 60 seconds).

B. UAM

UAM refers to an identity management solution. It is used by enterprises to manage digital identities and provide secure access to users across multiple devices and applications, both in the cloud and on the premises. UAM originates from

SSO (single sign-on) technology [2]. With SSO technology, employees only need to log in once to access all the mutually trusted services and resources within the enterprise. UAM extends SSO technology from within the enterprise to the entire Internet and provides a single platform from which IT personnel can manage access across a diverse set of users, devices, and applications, whether on the premises or in the cloud. Some organizations, including ITU-T, the Liberty Alliance, and MWS, are studying UAM standards and forming frameworks such as OMA and OpenID. For sites that support OpenID, you do not need to remember common authentication markers like a username and password; just enter your registered OpenID identifier and password and it will redirect you to the OpenID service site. Once you pass the authentication, you return to the original site and will have successfully logged in. The workflow of OAuth2.0 is similar to that of OpenID; the difference is that in OAuth2.0, the IDP is responsible for issuing four tokens, the authorization code, as well as the implicit password and client credentials, to third-party applications rather than the method of URL redirection. At present, OAuth2.0 is a popular third-party authorization and authentication framework that can be applied for UAM purposes [3], and it has been introduced into MSN, LinkedIn, Facebook, Twitter, WeChat, QQ, Alipay, etc.

In general, an UAM system consists of at least three parties: users, SPs (service providers), and IDPs (identity providers). An IDP accepts the user's identity registration request and verifies its validity; in addition, the IDP accepts authentication requests from SPs, which provide application services for users.

An identity authentication process in UAM is performed as follows:

1) The user requests services or resources from an SP.
2) The SP asks the user to provide identity information and the corresponding IDP address.
3) The user submits identity information and the IDP address to the SP.
4) The SP asks the IDP to authenticate the identity of the user.
5) After receiving the IDP's response, the SP decides to provide corresponding services or resources to the user as per the authentication results.

C. INTEL SGX

Intel SGX is a set of processor extensions for establishing a protected execution environment inside an application [7]. The following SGX technologies are used in this paper.

1) MEMORY ISOLATION

SGX allows user-level as well as operating system code to define private regions of memory, called enclaves, whose contents are protected within the boundary that cannot be either read or saved by any process outside the enclave itself, even the processes running at higher privilege levels.

2) SEALING

When the enclave process exits, the enclave is destroyed and any data that are secured within the enclave are lost. If the data are meant to be reused later, the enclave must make special arrangements to store the data outside the enclave. SGX provides the *sgx_seal_data* function, which retrieves a key unique to the enclave or the developer and uses that key to encrypt the input data buffer to preserve secret data. If needed, the sealed data blob can be unsealed by future instantiations of the enclave through the *sgx_unseal_data* function [8].

3) ECALL/OCALL

The programming model of an Intel SGX application differs from that of a typical application in that it requires the developer to repartition the program as per the official development manual, with each application divided into trusted and untrusted parts. Cross-domain function interfaces are defined and declared in the EDL (enclave define language) file, where the functions called by untrusted part and used to access data inside enclaves are declared as ECALL (enclave call); otherwise, to access an external resources, such as file system, network, or clock, the enclave must exit to the untrusted zone and performs a reverse context switch, those functions are declared as OCALL (outside call).

4) REMOTE ATTESTATION

Remote attestation convinces a remote party to have confidence that the intended software is securely running within an enclave on an SGX-enabled platform by producing an assertion. Remote attestation must rely on the Intel Attestation Service (IAS) to provide support for linkable signatures and attestation verification [9].

III. RELATED WORK

A. UAM IMPLEMENTATION AND APPLICATIONS

Many authentication standards and protocols have been specified in the last few years implementing UAM, such as OpenID (OIDF), which provides a way to prove that an user controls a specific identifier, OAuth (IETF), which focuses on managing access delegation, and OpenID Connect (OIDC), which extends OpenID to solve authentication besides authorization [10]. UAM idea can be connected with specific scenarios, such as power enterprises [11], campuses [12], hotels [13]. In general, the data integrity and privacy preservation of UAM are guaranteed through SSL/TLS [14], and the design of the identity database follows the LDAP protocol [15]. It is with regret that these applications are trapped in the existing UAM framework, namely, the SPs are asked to sign an agreement with an IDP, which fails to address mutual trust concerns between the SPs and IDP.

B. SECURITY ISSUES IN UAM AND OTPs

There are some works focused on presenting security issues of UAM and OTPs. The works can be divided into two groups.

The first group emphasizes the vulnerabilities and possible attack patterns, including (1) a client attack that the malicious adversary tries to violate the confidentiality and integrity of applications by memory dumping or DMA operation [16], (2) a phishing attack that redirects users to a malicious replica of an IDP or SP website [17]; For example, Cross-Site Request Forgery (CSRF) tricks a user into loading a page that contains a malicious request, which embeds the attack URL in an HTML construct, that could disrupt the integrity of the victim's session with a website [18], (3) an MITM attack that a middleman, who could perform two distinct Diffie-Hellman key exchanges with each party, masquerades as the IDP to sign authentication assertions or impersonates the users on the SP [19], (4) a replay attack that exploits the lack of assertion nonce checking by SPs [20], and (5) a DoS attack that attempts to exhaust the computational resources of SPs and IDPs [21].

For OTPs, they are not immune to agent attacks in addition to the foregoing attacks. Often, this happens when the OTP device's battery is low that the clock is unsynchronized. Worse, the device may be lost or stolen. Usually, there are methods to bypass the protection, for instance, via SMS or email, thereby providing an opportunity for attackers to exploit the loopholes [22].

The second group endeavored to perform formal security analysis and threat modelling, and utilized these models to find new weaknesses [23]–[25]. The formal analysis methods for security protocols used in UAM can be divided into two categories: One is the symbolic analysis [26], which regards the cryptographic system used in the protocol as a perfect loophole-free black box, and analyzes and proves the security protocols on this basis [27]; The other is the computational method, which assumes that the cryptographic system used by the protocol is not perfect. Crypto Verif is the first automated verification tool based on computational method [28]. The IETF even published a thorough threat model of OAuth2.0 in 2013.

C. EXISTING DEFENSE TECHNIQUES

(1) To resist client attacks, it is necessary to ensure that the client computer environment is trusted and no hacker programs whatsoever are running. This environment does not always withstand attacks even with firewall and antivirus programs installed; for example, malware such as keyboard hooks can embezzle secret information such as passwords entered by the keyboard [16]. The following two methods are considered helpful for preventing keyboard hooking: first, secure password box technology can be used. The secure password box works at both the kernel layer and application layer. Disabling scanning of port 0×60 or adding distractions at port 0×60 are some countermeasures at the kernel layer. In addition, modifying the address of the IDT interruption program can also bypass the keylogger at the kernel layer, and instead, bypassing is realized through encryption at the application layer [29]. Second, virtual keyboard technology can be used. Applications adopting virtual keyboard technology simulate real keyboard inputs by software and counterfeit a set of mapping keys at the application layer rather than the kernel layer. This method, however, is vulnerable to attacks through screen captures, and the keyboard inputs can be inferred based on the locations of the window and mouse click actions [30]. (2) To resist phishing attacks (viz. SSO CSRF), validating the HTTP Referer header to ensure the request in question was issued by an authorized source is a simple way, but this might be impractical since many web proxy servers suppress the Referer headers due to privacy concerns [31]. Another way is using machine learning models to detect and defend against phishing attacks [32]. There are also proposals to install phishing detection software in web browsers [33]–[35]. (3) As for the MITM attacks, the SSL/TLS protocol is considered as the first choice for most authentication schemes [36]. (4) D. Kreutz *et al.* built resilient and secure authentication and authorization infrastructures, called R-OpenID prototypes, by providing an architectural model and system design artifacts to defend against large-scale DDoS [37]. S. Qiu *et al.* even presented an improved authentication scheme claiming that "it can resist all known attacks" [38].

To deal with the agent attacks on OTPs, extending two-factor authentication to three-factor authentication is an easy way. The three-factor AKA protocol for mobile lightweight devices is just one of the examples offered by [39].

It must be said that some techniques are deemed too complex and expensive for practical use. While the one-fit-all scheme that resists every possible attack has not been put forward yet, an UAM scheme that is omnipotent in terms of security, if possible, would thus be favored.

IV. THREAT MODEL AND SECRUITY CLAIMS

Recently, many investigations have been conducted on the security of well-established protocols and standards, and it turns out that classical cryptography just owns robustness in the quondam but not any longer [17], and it is believed that ambiguous security properties and the lack of clear threat models are blamed. In other words, it is of prime importance to clearly define the threat model and security claims. In this regard, this section presents the threat model and security claims.

A. THREAT MODEL

This research assumes that all the softwares including privileged operation systems and hypervisors in the IDP, the SP and the user client are untrusted. A malicious adversary tries to violate the confidentiality and integrity of user applications by memory dumping or DMA operation. The intruder can also subvert the untrusted part of an authentication module (the trusted part being protected by SGX). With these contexts as premise, the desired approach requires that the following threats be addressed:

1) Client attacks. If the user's computer is hacked with Trojan horses, spyware, and so forth, the hacker program may intercept the dynamic password entered

by the user through keyboard hooking; or steal application data through memory dumping.

2) Man-in-the-middle (MITM) attacks. The session hijacker can observe the conversation between an user client and an IDP by using a sniffer; this person may, on the one hand, block the user from being able to log in by means of creating the illusion that the network is disconnected or timed out, or, on the other hand, impersonate the user to access the service.
3) Phishing attacks. Attackers set up phishing sites redirecting the user to an untrusted network [40].
4) Replay attacks. To launch such attack, the adversary needs to sniff some of the involved communication channels to obtain login credentials to form an access token, then re-use legitimate token at the target SP unable to detect the replay (because tokens have not expired yet or because this expiration is not properly checked) or at other SPs not validating the tokens' audience.
5) Denial of service (DoS) Attacks. An DoS attack is an attack meant to shut down IDP machine or network, thwarting the authentication service or making it inaccessible to its intended users.

B. SECURITY CLAIMS

The following terms are used to describe the security properties of our approach. For each of them we provide the respective countermeasures.

1) Integrity and authenticity protections. Integrity provides a means to detect unauthorized modification, and authenticity allows the receiver to verify if the sender is who it claims to be. We satisfy integrity by remote attestation of Intel SGX letting the source codes of intended software released to the public that any modifications of such software could render an attestation failure; we achieve authenticity by using secure channels such as TLS handshake.
2) Confidentiality protections. This refers to the capability of protecting the confidentiality of sensitive data and operations. Intel SGX helps establishing private regions of memory in which application data cannot be read or saved by any process outside the enclaves, thus serves for our confidentiality purpose; in addition, anti-hook mechanism designed in this paper can protect client from keyboard hooks installed by local malwares.
3) Robustness. Nowadays, availability is a first class requirement for any IT infrastructure, it requires the system being accessible in face of DoS attacks or unforeseen damages. To that end, OTP and blocking-invocation are used to avoid system crash due to intentional attacks.

V. THE PROPOSED SCHEME

A. ROLE SETTING

An SGX-based unified access management system (we name it as "SGX-UAM") involves four types of roles: client (user), identity provider (IDP), service provider (SP) and IAS, as shown in Figure 1. The "SP" herein refers to the third-party website that the user want to access, viz. the "SP" role in the UAM terminology, rather than the "SP" role in Intel SGX remote attestation, who is generally the advocator of the attestation.

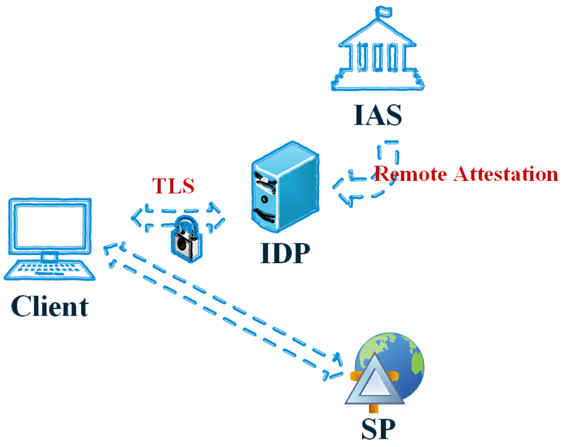


FIGURE 1. Role setting of SGX-UAM.

B. SOFTWARE PROCESS

A complete SGX-UAM software process includes four phases: registration, login credentials storage, OTP authentication, and login credentials retrieval.

The notations and their meanings are enlisted in Table 1.

TABLE 1. Notations.

Notation	Description
σ	one-time password (OTP)
$\prod$	HMAC function
ε	OTP key
τ	interception function of OTP
I	user identifier
M	user masterkey
h_1	hash value of user identifier
h_2	hash value of user masterkey
$\boldsymbol{H}$	MD5 function
s	SGX seal key
χ	exclusive-OR result of h_1 and h_2
κ	secret key
f	dynamic factor
Σ	signature algorithm
$\dagger$	authentication result
Γ	signature of authentication result of user I

1) REGISTRATION PHASE

Before registration, the IDP and client are required to finish bidirectional remote attestation with IAS participation (a procedure to promote mutual trust).

Then, the user manually enters the identifier, denoted as I, and masterkey, denoted as M, as the registration information, where I is usually the frequently used email address of the user, and M is created by the user. After that, the client enclave computes the hash values of I and M, denoted as h_1 and h_2 respectively. The MD5 function is determined as the hash function (marked as $\boldsymbol{H}$). The client is asked to sends h_1 and h_2 to the IDP for future uses. Followingly, the IDP will check whether h_1 exists; if not, it saves them to the database; otherwise, it returns an error message.

The IDP enclave is suggested to seal h_1 and h_2 to the hard disk for retrieval in case a power outage or system crash causing the enclave destroyed unpredictably could happen.

The registration flow is shown in Figure 2.

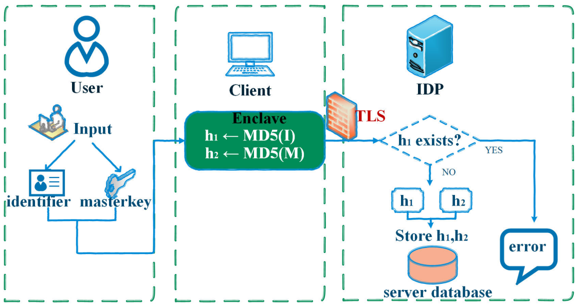


FIGURE 2. Registration flow.

2) LOGIN CREDENTIALS STORAGE PHASE

The user enters the login credentials that needs to be managed, including an URL, an username and a password.

The client enclave performs the bitwise exclusive-OR operation on h_1 and h_2; it calls the *sgx_get_key* function, which is a wrapper for the EGETKEY instruction, to generate a 128-bit seal key s. The s and exclusive-OR result χ are concatenated, and the MD5 function is carried out on the concatenated string to obtain the final secret key κ. The pseudocode is presented in Algorithm 1.

Algorithm 1 Secret Key Generation Algorithm

```
Input: identifier $I$, masterkey $M$;
Output: Secret key $\kappa$;
1: function SecretKeyGeneration($I, M$)
2:     $h_1 \leftarrow \boldsymbol{H}(I)$;
3:     $h_2 \leftarrow \boldsymbol{H}(M)$;
4:     $\chi \leftarrow h_1 \oplus h_2$;
5:     $s \leftarrow sgx\_get\_key()$;
6:     $\kappa \leftarrow \boldsymbol{H}(s \vee \chi)$;
7:     return $\kappa$;
8: end function
```

The client enclave then encrypts the login credentials with SM4 as the encryption algorithm and κ as the key and uploads the ciphertext to the IDP.

The login credentials storage flow is shown in Figure 3.

3) OTP AUTHENTICATION PHASE

In the initial login stage, the client requests time synchronization from the IDP and then takes the current timestamp as the dynamic factor f, which explains the real-time performance of the OTP.

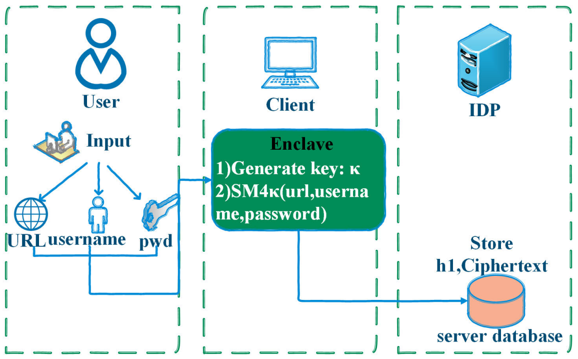


FIGURE 3. Login credentials storage flow.

The IDP starts the timer, and the dynamic factor expires after 60 seconds.

In the client enclave, the string obtained by stitching the two together is input into the $KGEN(\cdot)$ function to obtain the OTP key (denoted as ε) required for this login. Then the function takes the OTP key and the dynamic factor received from the IDP as input and generates the OTP (denoted as σ). The pseudocode of OTP generation algorithm is shown as Algorithm 2.

Algorithm 2 OTP Generation Algorithm

```
Input: identifier hash $h_1$, masterkey $h_2$, dynamic factor $f$;
Output: one-time password $\sigma$;
1: function OTPGeneration($h_1, h_2, f$)
2:     $\varepsilon \leftarrow$KGEN($h_1 \vee h_2$);
3:     $\sigma \leftarrow$OTPGEN($\varepsilon, f$);
4:     return $\sigma$;
5: end function
```

A viable $KGEN(\cdot)$ can be the MD5 function, viz.

$$\varepsilon = \boldsymbol{H}(h_1 \vee h_2). \tag{2}$$

and a feasible $OTPGEN(\cdot)$ function can be HMAC function, viz.

$$\sigma = \tau(\prod(\varepsilon, f)). \tag{3}$$

Then, we send the generated OTP along with h_1 to the IDP for comparison. The IDP finds the corresponding h_2 value by indexing h_1 in local database, then performs an exactly same generation process as the client to obtain an OTP, and finally compares it with that received from the client. If the OTPs are consistent and the f has not expired, the user is authenticated successfully; otherwise, an error message is returned. The IDP signs the authentication result with a private key.

$$\Gamma = \Sigma_{.sig}(I, \dagger). \tag{4}$$

The OTP authentication process is shown in Figure 4.

4) LOGIN CREDENTIALS RETRIEVAL PHASE

login credentials retrieval proceeds as follows (see Figure 5):

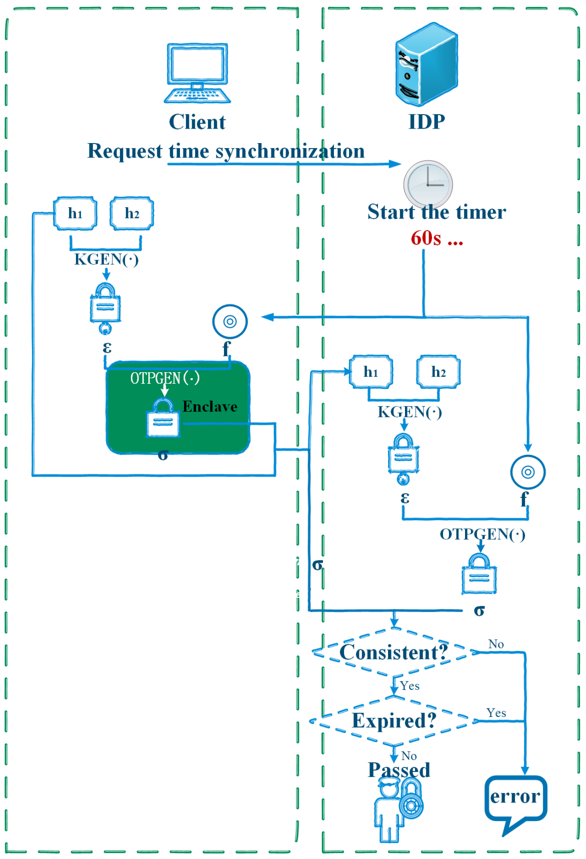


FIGURE 4. OTP authentication flow.

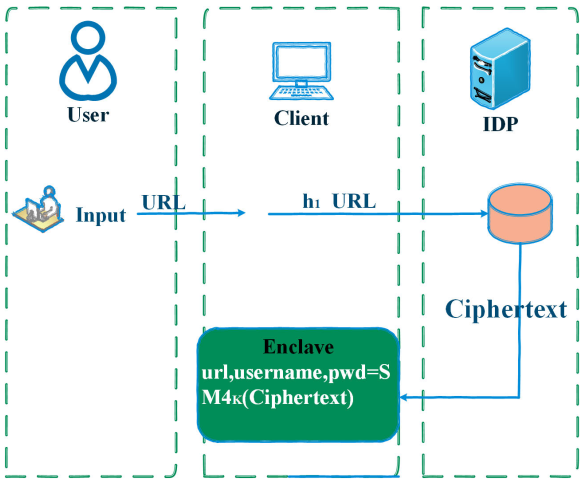


FIGURE 5. Login credentials retrieval flow.

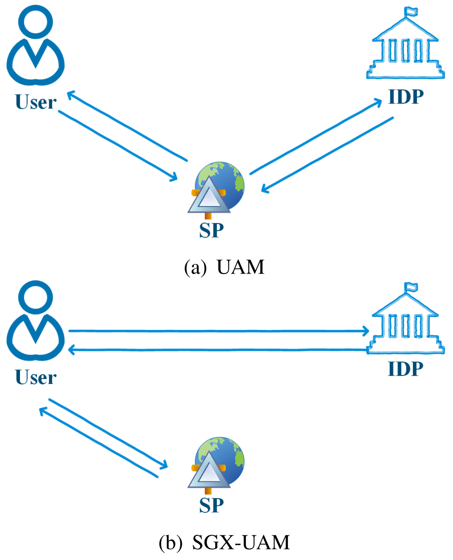


FIGURE 6. The difference between UAM and SGX-UAM.

1) The client sends h_1 and URL to the IDP.
2) The IDP looks up the encrypted login credentials in local database as per h_1 and URL and sends them back if found.
3) The client provides the login credentials to the user after decrypting the ciphertext within the SGX enclave.

When beginning to log in, the client sends the OTP authentication result signed by the IDP to the website, the website then verifies the signature of the IDP; if verified, the identity is authenticated, and the login attempt is allowed; otherwise, the login attempt is in vain.

C. THE DIFFERENCE BETWEEN SGX-UAM AND UAM

Generally, UAM requires an SP and an IDP to establish authorization agreements, such as OAuth2.0. As some SPs are influential and own robust identity databases, they are not inclined to entrust all authentication businesses to third-party operators. SGX-UAM differs from UAM in that we design the authentication process between the user and the IDP instead of between the SP and the IDP, thus forcing the user to shoulder the burden rather than the SP (see Figure 6). Considering that the user is often the resource-constrained entity out of the two, it is actually a tough decision that the system has to sacrifice a little performance to accommodate the commercial concerns of SPs. Fortunately, influential SPs usually have established mature authentication mechanisms, so there is no need to resort to third-party authentication services; and SGX-UAM scheme can preserve users' privacy because all login credentials are stored in ciphertext form in IDP. In Section VII, we will check whether the performance loss is unacceptable compared with other UAM schemes to assess our decision.

VI. SECURITY ANALYSIS

In what follows, we evaluate the proposed scheme in terms of a security analysis.

A. RESISTANCE TO CLIENT ATTACKS

Resistance to client attacks is mainly realized by ensuring the security of the input and memory.

1) INPUT SECURITY

It is possible for a client to leak confidential information due to the existence of keyboard hooks. Hooks can intercept a user input and possibly tamper with it before it reaches the destination window. According to reference [34], keyboard hooks were realized through two methods. The first uses several Windows functions such as *SetWindowsHookEx*, *CallNextHookEx*, and *CopyMemory* to obtain the actions performed by the keyboard and determine the exact key inferred from the virtual key table. The second adopts the *GetAsyncKeyState* function.

In this regard, we attempt to prevent such attacks through "shuffle mapping" when keyboard events are triggered. Here are the details of how "shuffle mapping" works.

Suppose there are N global hook processes. If our hook process is P_N and that of the attacker is P_{N-1}, then unsurprisingly, P_N would have the initial right to process messages and pass them into the SGX enclave via ECALL. Assuming that there are Q keys on the keyboard, correspondingly, the enclave generates Q random numbers, $r_1, r_2, \cdots, r_Q$, as alternatives to the key values (see Table 2). The mapped keys are then substituted for the real keys in the system message queue via OCALL. Figure 7 shows the SGX-UAM keyboard hooking mechanism.

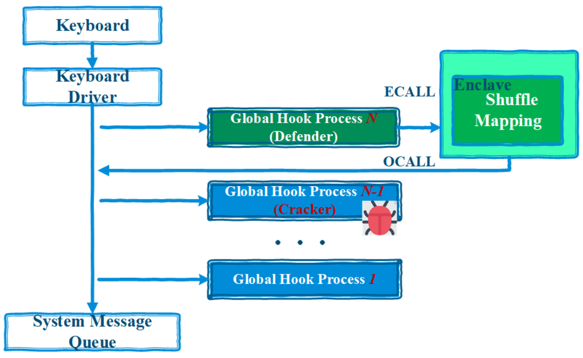


FIGURE 7. SGX-UAM keyboard hook mechanism.

Nevertheless, how on earth does SGX-UAM guarantee the right to pre-emptively process a message?

One reason is that SGX-UAM contains two special Windows message hooks: *WH_DEBUG* and *WH_KEYBOARD_LL*, where *WH_DEBUG* covers most event functions, such as *WH_KEYBOARD* and *WH_MOUSE*. The *WH_DEBUG* hook always takes precedence over general keyboard hooks.

TABLE 2. Mapping table of key values.

Key name	Value	Description	Mapping value
vbKeyTab	9	Tab key	r_1
vbKeyShift	16	Shift key	r_2
vbKeyEscape	27	Esc key	r_3
vbKeyHome	36	Home key	r_4
vbKeyUp	38	UP Arrow key	r_5
vbKeyPrint	42	PrScrn key	r_6
vbKeyA	65	A key	r_7
...	...	...	...

Another secret lies in our periodic hooking mechanism. In Windows, if the mechanism is loaded with more than one hook of the same type, the hooks are executed in the order of "first installed, last executed". If the cracker's hook is installed after our hook, it executes first, and we cannot obtain messages ahead of the cracker. To avoid this, we set a timer to unhook our keyboard hooks every few seconds and then hook again immediately. According to the principle of "last installed, first executed", it is highly possible that our keyboard hooks will execute ahead of those of the cracker. Only once they hook successfully can they prevent the cracker from stealing the correct password— for instance, if SGX-UAM hooks every 2 seconds, and the cracker hooks every 0.5 seconds, then it may occur that just after the cracker finishes hooking, SGX-UAM hooks immediately. In the next 0.49 seconds, as long as the user presses the keyboard, the cracker's task is considered to have failed (see Figure 8).

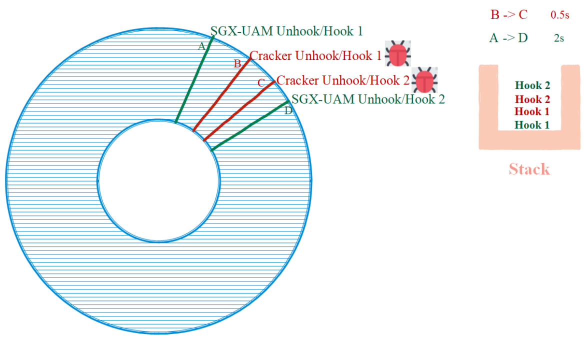


FIGURE 8. SGX-UAM periodic hooking.

The pseudocode of SGX-UAM hooking is shown as Algorithm 3.

Algorithm 3 SGX-UAM Keyboard Hook Algorithm

```
Input: Keyboard input $m = \{c_i\}, i = 1, 2, \cdots;$
Output: Shuffled keyboard input $m'$;
 1: function OnTimer( )
 2:     $UnhookWindowsHookEx()$;
 3:     $SetWindowsHookEx()$;
 4: end function
 5: function OnGetChar($c_i$)
 6:     $SGX :: ECALL(c_i)$;
 7:     $\{r_1, r_2, \cdots, r_K\} \leftarrow Enclave :: sgx\_read\_rand()$;
 8:     $c'_i \leftarrow Enclave :: GenerateMapping(c_i)$;
 9:     $SGX :: OCALL(c'_i)$;
10: end function
11: $m'.Add(c'_i)$;
12: return $m'$;
```

2) MEMORY SECURITY

The code and data in the SGX enclave utilize a threat model in which the enclave is trusted but no process outside it can be trusted (including the operating system itself and any hypervisor software); therefore, all of these processes are treated as potentially hostile.

SGX-UAM puts OTP generation and the encryption and decryption of login credentials into enclave to prevent memory from being snooped on. The key for encrypting/decrypting login credentials are dynamically generated within the enclave through *EGETKEY* instruction (essentially CPU-bound), and any attempts to access the key are trapped outside the enclave.

In addition, the OTP generation process depends on the hash value of user's masterkey h_2, which is suggested to be sealed on the hard disk and can only be unsealed by the enclave, thus guaranteeing that h_2 can never be stolen.

B. RESISTANCE TO MITM ATTACKS

The attackers often sniff or even tamper with the data by intercepting network packets, without the two parties in the communication being fully aware of it [49]. This kind of attack cannot succeed since the generation of an OTP requires h_2, which cannot be obtained by the middleman. In addition, even if the middleman intercepts the generated OTP, the final login credentials are transmitted in ciphertext form and cannot be decrypted outside the client enclave.

C. RESISTANCE TO PHISHing ATTACKS

If the SP URL is not recorded in the IDP's database, the IDP refuses to provide the authentication service. In other words, phishing sites or unregistered sites are inaccessible. Therefore, the efforts to launch a phishing attack are unavailing.

D. RESISTANCE TO REPLAY ATTACKS

A replay attack occurs when a cybercriminal eavesdrops on a secure network communication, intercepts it, and then fraudulently delays or resends it to misdirect the receiver into doing what the hacker wants. Replay attacks happen, all too often, in an UAM system that the eavesdroppers intercept login credentials, user activity, computer and browser specs, and passwords at will.

SGX-UAM adopts OTPs with time-related dynamic factors to resist replay attacks. Considering inevitable network latency, the dynamic factors must remain valid for a duration (e.g., 60 seconds) between the authenticator (IDP) and the authenticatee (user). For subsequent authentications to work, the clocks of the authenticatee and the authenticator need to be roughly synchronized again. So even if an attacker eavesdrops the communication and repeat to send, she could hardly succeed in getting login credentials from IDP.

E. RESISTANCE TO DOS ATTACKS

In SGX-UAM, authentication requests from a single supplicant is handled in a blocking way, that means if an adversary plans to launch DoS attacks, enormous amounts of user accounts are needed. This sounds rather contrived since our policy is that user account has to be linked with a valid email address.

VII. EXPERIMENTS

In what follows, we evaluate the performance of SGX-UAM from the aspects of OTP authentication time, response time, throughput, and resource depletion, and evaluate the security of SGX-UAM by seeing system behaviour under client attacks, MITM attacks, phishing attacks, replay attacks and DoS attacks.

The experimental setup creates a primitive SGX-UAM prototype that implements four main phases, viz. registration, login credentials storage, OTP authentication, and login credentials retrieval. The software involves three roles including an IDP, an SP and an user/client, where the IDP component is mounted on a Sugon X745-G30 4U server with the hardware configuration of SGX-enabled Intel® Core E3-1240L v5 processor, 128GB DDR RAM, 2TB SSD, 4TB SAS hard disk and dual Gigabit Ethernet NICs; the SP offers web service with Django (a python web framework) deployed, and the SP component is mounted on a virtual machine (VM) in the aforementioned server allocated 8 GB RAM and 100 GB hard disk; the user component is mounted on a PC equipped with SGX-enabled Intel® Core I5-2430m, 8 GB RAM and 500 GB SAS hard disk. The simulation is conducted in a LAN environment. The role setting and corresponding hardware configuration in this experiment are shown in Table 3.

TABLE 3. Role setting and corresponding hardware configuration in the experiments.

Role	Hardware attributes	Machine type
User	Intel ® Core I5-2430m, 8 GB RAM and 500 GB SAS hard disk	PC
IDP	Intel® Core E3-1240L v5, 4*32 GB DDR RAM, 2 TB SSD and 4 TB SAS hard disk	Server
SP	Intel® Core E3-1240L v5, 8GB RAM and 100 GB hard disk	VM hosted on server

A. PERFORMANCE EVAUATION

1) OTP AUTHENTICATION TIME

Native OTPs (including HOTP and TOTP) are selected as the benchmarks. The OTPs generated in SGX-UAM utilize Intel SGX to ensure confidentiality with relevant SGX library functions enlisted in Table 4.

TABLE 4. The SGX library functions used in OTP gengeration phase of SGX-UAM.

Functionality	SGX library function
Time Synchronization	*sgx_get_trusted_time*
Event Counter	*sgx_create_monotonic_counter* *sgx_increment_monotonic_counter* *sgx_read_monotonic_counter*
KGEN(·)	*sgx_sha256_msg*
OTPGEN(·)	*sgx_rijndael128_cmac_msg* *sgx_hmac_sha256_msg*

Figure 9 gives the average OTP generation time of SGX-UAM compared with native OTPs. A total of 10 tests were performed, with the mean generation time expressed by bars and the vertical line above each bar denoting the standard deviation. We see that

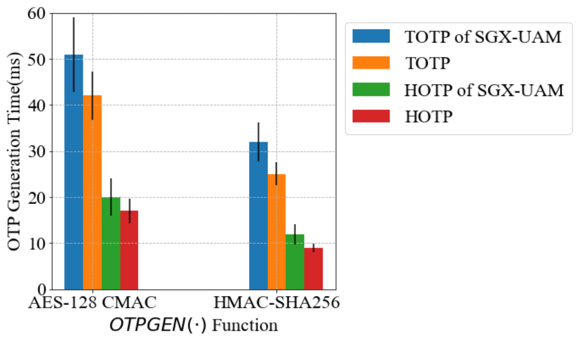


FIGURE 9. Average OTP generation time of SGX-UAM compared with native OTPs. A total of 10 tests were performed, with the mean value expressed by bars and the vertical line above each bar denoting the standard deviation.

1) The OTPs (TOTP and HOTP) generation in SGX-UAM takes more time than native OTPs. This is because the use of SGX does impose a penalty on performance.
2) In terms of the selection of *OTPGEN*(·) functions, HMAC-SHA256 is considered better than AES-128 CMAC for its greater generation speed. We owe the result to the fact that hash operation is generally faster than block ciphering.

2) RESPONSE TIME

We implemented a complete authentication process embracing all four phases of SGX-UAM, and we also implemented OpenID and OAuth2.0 as the benchmarks. To prevent the doubt about the quality of the implementation, we quote two studies in which similar experiments were conducted. Table 5 gives the summary of relevant studies and our implementations in terms of the response time for one authentication request. It can be seen that OpenID, OAuth2.0 and SGX-UAM achieve almost the same performance despite the differences in hardware configuration. Howbeit the response time obtained in reference [41] is longer than the others, it is not enough to overturn the aforesaid conclusion given that they are still in the same order of magnitude.

3) THROUGHPUT

The throughput of SGX-UAM was measured using 2-20 simultaneous supplicants. Each supplicant was configured to execute 10,000 sequential authentications using the same credentials. Furthermore, each authentication requires exactly ten packets. A thread pool is designed for accommodating simultaneous supplicants, but no new threads would be activated for sequential authentications requested from one supplicant, that means every request launched from one supplicant or user connection is handled in a blocking way.

Figure 10 shows the turnaround time for authentication requests from 2-20 simultaneous supplicants/connections. It can be seen that:

1) The turnaround time remains stable with the increase of user connections for all three schemes we implemented.

TABLE 5. Summary of relevant studies and our implementations in terms of the response time for one authentication request.

Ref. [1]	Spec. [2]	Time [3]	Env.[4]
D. Kreutz et al.(2016) [37]	OpenID	100 ms	m3.large (Amazon Web Services, Inc., 2014) computing nodes (running Ubuntu Server 14.04 LTS)
K. Chaturvedi et al. (2019) [41]	OAuth2.0	428 ms	Linux OS with 4 virtual CPUs, 8 GB RAM hosted on a VMware ESXI 5.1.0 Server with 8 CPUs, 2 Processor Sockets, 4 Cores per Socket running with 2.27 GHz.
This paper	OpenID OAuth2.0 SGX-UAM	110 ms 108 ms 109 ms	Sugon X745-G30 4U server with Intel® Core E3-1240L v5, 128 GB RAM running CentOS 7

[1] Ref.: references
[2] Spec.: specified UAM implementations
[3] Time: response time
[4] Env.: emulation environment

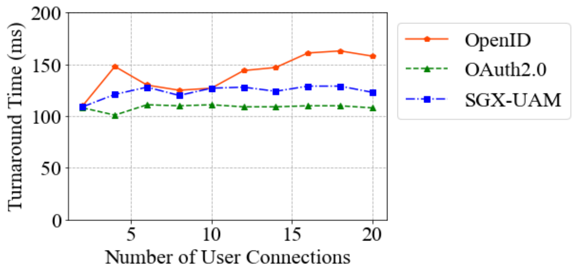


FIGURE 10. Turnaround time for authentications from 2-20 simultaneous supplicants/connections.

2) SGX-UAM achieves a mid-level performance out of the three.

Figure 11 shows the turnaround time for processing 1-10,000 sequential authentication requested from one supplicant. Those requests are processed in blocking-invocation style. We see that:

1) As the number of sequential authentication requests increase, the turnaround time increases at a decreasing rate.
2) OpenID consumes the most time in handling the same amount of sequential authentication requests, followed by SGX-UAM, and then OAuth2.0.

4) RESOURCE DEPLETION

Further, we analyze the resource depletion of IDP, SP and user client in the throughput experiment. The targets include maximum memory footprint, CPU utilized percent, system calls, page faults and interrupts.

For IDP, the maximum memory footprint and CPU utilized percentage when handling 1, 100 and 10,000 sequential requests for one user connection are shown in Figure 12. It can be seen that:

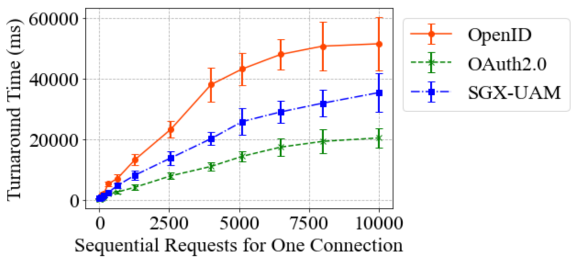


FIGURE 11. Average turnaround time for processing 1-10,000 sequential authentication requests for one supplicant/connection, each point in the plot is an average of 10 tests. The vertical line above each curve denoting the standard deviation.

TABLE 6. System calls, page faults and interrupts of IDP when handling sequential requests for one supplicant/connection.

Req.[1]	Spec.[2]	Sys.Call (k)[3]	PF (k)[4]	Intrp.[5]
	OpenID	247	60	654
1	OAuth2.0	120	2	77
	SGX-UAM	473	28	473
	OpenID	668	359	734
100	OAuth2.0	275	3	117
	SGX-UAM	1443	65	564
	OpenID	1844	423	956
10000	OAuth2.0	903	4	151
	SGX-UAM	5493	162	1033

[1] Req.: number of sequential requests
[2] Spec.: specified UAM implementation
[3] Sys. Calls: system calls
[4] PF: page faults
[5] Intrp: interrupts

1) For up to 10,000 sequential requests, the maximum memory footprint is only 153 MB, which is innocuous.
2) SGX-UAM has higher CPU usage than OpenID and OAuth2.0.

Considering that privileged software can manipulate the page tables of an enclave to observe a page-granularity trace of its code and data, this leakage channel would disappear if SGX enclaves serviced their own page faults. Therefore, page faults are also measured, and obviously the fewer page faults the better. Table 6 shows other metrics not shown in Figure 12, including system calls, page faults and interrupts. We see that:

1) SGX-UAM has fewer page faults and interrupts than OpenID though more system calls, means that it is more secure.
2) All metrics fluctuates with the number of requests.

The maximum memory footprint and CPU utilized percent of user client is shown in Figure 13. We see that:

1) SGX-UAM consumes the most memory and CPU compared with OpenID and OAuth2.0.
2) The maximum memory footprint of SGX-UAM is 37 MB when handling up to 10,000 requests, which can be considered acceptable.

B. SECURITY EVAUATION

Five kinds of concrete attacks are conducted to assess the security of SGX-UAM.

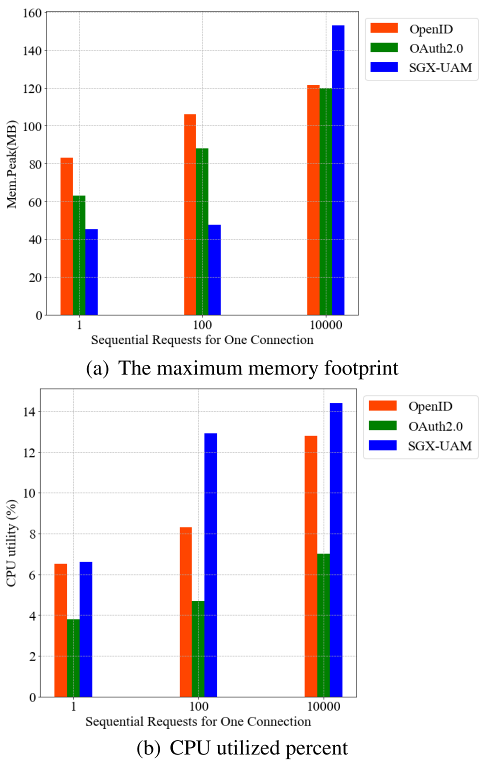
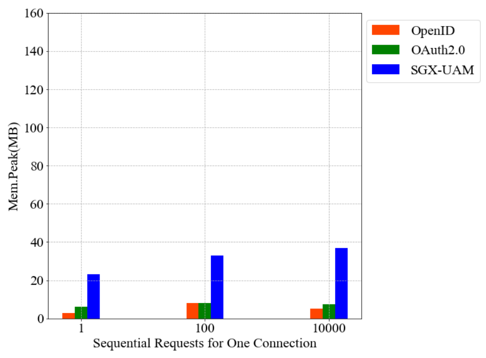


FIGURE 12. The maximum memory footprint and CPU utilized percent of IDP when handling 1, 100 and 10,000 sequential authentication requests for one supplicant/connection.

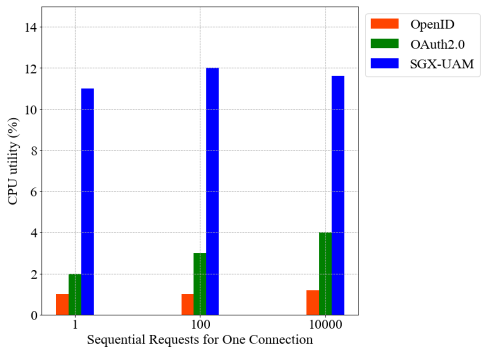
The first is a client attack. A keyboard hook program intending to get the keyboard action and determine the actual key is installed to carry out such attack. The steps including creating a dialog-based Qt application and implementing the interface of *KeyboardProc* (a system API in Microsoft Windows). SGX-UAM defends against it by "periodic hooking" with an interval of 500 milliseconds.

MITM attacks, phishing attacks, replay attacks and DoS attacks are conducted with *Zarp*. *Zarp* is a network attack tool that offers the following types of tests: Poisoners, DoS, Sniffers, Scanners, Services, Parameter, and Attacks. which open up the possibility for very complex attack scenarios. Concretely,

1) To launch MITM attacks, *Zarp* Sniffers, who act as the middleman, intercepts the generated OTP and the encrypted login credentials transmitted between the IDP and the user. The success of such attacks is measured on whether the logging is allowed for the "impersonator" finally.
2) To carry out phishing attacks, *Zarp* Poisoners hide malicious URL in HTML construct and poison the DNS cache records. The attacks are considered successful if the "phisher" tricks the users into providing login credentials by means of redirecting the URL.
3) To carry out replay attacks, *Zarp* Sniffers is eavesdropping on the conversation between IDP and the user, and keeps the OTP. After the interchange is over, *Zarp*



(a) The maximum memory footprint



(b) CPU utilized percent

FIGURE 13. The maximum memory footprint and CPU utilized percent of user client (PC device) when launching 1, 100 and 10,000 sequential authentication requests.

Sniffers connects to the IDP, when asked for proof of identity, *Zarp* Sniffers sends user's OTP read from the last session which user accepts, thus granting "eavesdropper" access. This attack is deemed successful if *Zarp* Sniffers pass the OTP authentication within the expiry time.

4) To start the TCP SYN DoS attacks, we'll flood port 8080 at the IP address 192.168.0.101 of the IDP, with 100,000 packets. If the delay to compete one authentication is up to 10 seconds, such DoS attack is considered successful.

The results of the foregoing attacks are enlisted in Table 7. It can be seen that

1) SGX-UAM performed best in anti-attack tests that keeping the odds of a successful attack stable at a lower level, compared with OpenID and OAuth2.0.
2) 5% client attacks, 7% replay attacks, and 3% DoS attacks reached their goals, which indicates that "periodic hooking" would capture most but not all malicious hooks; the 60-second expiry time of OTP gives eavesdropper a chance to deceive the IDP, but shortening the expiry period would fix the problem; the success rate of DoS attacks can drastically, though not always, be reduced, because requests processed synchronously are invoked in a blocking way.

Table 8 gives a summary of security properties of UAM schemes among relevant references. SGX-UAM can outdo

TABLE 7. Attacks results.

Type	Attempts	Success Rate		
		OpenID	OAuth2.0	SGX-UAM
Client attacks	100	88%	55%	5%
MITM attacks[1]	100	76%	83%	0%
Phishing attacks	100	73%	14%	0%
Relay attacks	100	10%	21%	7%
DoS attacks	200	35%	58%	3%

[1] Man-in-the-middle attacks.

TABLE 8. Security comparison of UAM schemes among relevant references.

Ref.[2]	Spec.[3]	Attack Type[1]				
		Client	MITM[4]	Phishing	Relay	DoS
Sun et al. [42]	OpenID	●	●	●	◑	×
Hu et al. [43]	OAuth	×	●	●	●	●
Yang et al. [44]	OAuth	●	●	●	●	×
Bansal et al. [23]	OAuth	×	●	●	×	×
Fett et al. [46]	OAuth	×	●	●	●	×
Yang et al. [47]	OAuth	×	●	●	●	×
Birrell et al. [21]	Many	×	●	●	●	●
Mainka et al. [48]	OAuth2.0	×	●	◑	◑	×
Werner et al. [49]	OAuth2.0	×	×	◑	◑	×
This paper	SGX-UAM	◑	○	○	◑	◑

[1] ○: completely immune; ◑: partially immune; ●: completely vulnerable; ×: not mentioned.
[2] Ref.: references.
[3] Spec.: specified UAM implementations.
[4] MITM: Man-in-the-middle attacks.

OpenID, OAuth and OAuth2.0 when it comes to the ability of resisting attacks, especially MITM attacks and phishing attacks. This is in line with our simulation results in Table 7.

VIII. CONCLUSION

This paper aims to design a highly secure unified access management system by using Intel SGX, we name it "SGX-UAM". It outperforms most UAM implementations such as OpenID and OAuth2.0 in anti-attack abilities. Specifically, it resists most client attacks, MITM attacks, phishing attacks, most replay attacks and most DoS attacks to which the generic UAM implementations are vulnerable, and we confirmed this by experiments. SGX-UAM also differs from generic UAM in that it assigns the burden of identity authentication to the users rather than the SPs, which not only relieves the security concerns of SPs but also protects users' privacy, and the performance loss is acceptable to be worth the try, in retrospect.

The simulation results show that SGX-UAM seems poor somehow in relation to native OTP schemes such as TOTP and HOTP, and we owe this inefficiency to the fact that SGX itself does impose a performance penalty; but from another perspective, we thus gain the benefit that we no longer need an additional hardware because CPU-bound SGX can guarantee the uniqueness of the device; in contrast, native OTPs usually require the user to maintain a synchronizer or event counter, which are often achieved via sepcial hardware. SGX-UAM consumes almost the same time with OpenID and OAuth2.0 to complete one login request, and it achieves a mid-level performance out of the three when handling vast login requests. The memory usage and CPU utilized percentage of the IDP for SGX-UAM is innocuous, and that of the client is acceptable.

Considering that Intel® SGX is essential for SGX-UAM, the cost and overhead are bound to increase. Future studies should target on further reducing the burden of the client. Besides, more ideas about how to resist replay attacks and DoS attacks should be presented.

ACKNOWLEDGMENT

The authors would like to thanks *paolostivanin* (nickname) from GitHub for offering the C-implemented code of OTP generation at https://github.com/paolostivanin/libcotp, and *openid* (nickname) and *udacity* (nickname) for providing Python implementations of OpenID and OAuth2.0 (accessible at https://github.com/openid/python-openid and https://github.com/udacity/OAuth2.0).

REFERENCES

[1] L. Liu and J. Wang, "Unified identity management and application analysis," *Telcommun. Tech.*, vol. 6, no. 4, pp. 86–90, 2009.

[2] H. Z. Gao, "Research on single sign-on technology," M.S. thesis, School Softw., Beijing Univ. Posts Telecommun., Beijing, China, 2006.

[3] B. Zhu, "Research on the status quo of domestic application of authorized login based on OAuth 2.0 protocol," *Mod. Inf. Technol.*, vol. 3, no. 20, pp. 151–154, 2009, doi: 10.19850/j.cnki.2096-4706.2019.20.051.

[4] R. Wang, S. Chen, and X. Wang, "Signing me onto your accounts through Facebook and Google: A traffic-guided security study of commercially deployed single-sign-on Web services," in *Proc. IEEE Symp. Secur. Privacy*, Oakland, CA, USA, May 2012, pp. 365–379.

[5] H. Frank, N. David, B. Mihir, and R. Ohad, "HOTP: An HMAC-based one-time password algorithm," Internet Eng. Task Force, Wilmington, DE, USA, Tech. Rep., 2005. [Online]. Available: https://tools.ietf.org/html/rfc4226#section-5.3

[6] D. M'Raihi, S. Machani, M. Pei, and J. Rydell, *TOTP: Time-Based One-Time Password Algorithm*, document RFC 6238, Internet Engineering Task Force, Wilmington, DE, USA, 2011. [Online]. Available: https://tools.ietf.org/html/rfc6238

[7] Intel. (2018). *Intel Software Guard Extensions*. [Online]. Available: https://software.intel.com/en-us/sgx

[8] I. Anati, S. Gueron, S. Johnson, and V. Scarlata, "Innovative technology for CPU based attestation and sealing," in *Proc. 2nd Int. Workshop Hardw. Archit. Support Secur. Privacy*, 2013, p. 7.

[9] J. Wang, C.-Y. Fan, Y.-Q. Cheng, B. Zhao, T. Wei, F. Yan, H.-G. Zhang, and J. Ma, "Analysis and research on SGX technology," *J. Softw.*, vol. 9, no. 29, pp. 2778–2798, 2018.

[10] J. Navas and M. Beltrán, "Understanding and mitigating OpenID connect threats," *Comput. Secur.*, vol. 84, pp. 1–16, Jul. 2019, doi: 10.1016/j.cose.2019.03.003.

[11] X. J. Chen, "The realization of PKI/CA unified identity authentication," *Tech. Innov. Appl.*, vol. 8, no. 31, pp. 55–56, 2009.

[12] J. Hua and X. Qu, "Uniform identity authentication system based on LDAP protocol," *Intell. Comput. Appl.*, vol. 9, no. 3, pp. 129–134, 2019, doi: 2095-2163(2019)03-0129-04.

[13] W. M. Lim, P.-L. Teh, P. K. Ahmed, S.-N. Cheong, H.-C. Ling, and W.-J. Yap, "Going keyless for a seamless experience: Insights from a unified hotel access control system," *Int. J. Hospitality Manage.*, vol. 75, pp. 105–115, Sep. 2018.

[14] P. Yang and M. H. Yin, "Realization of unified identity authentication platform with SSL VPN technology," *Cyberspace Secur.*, vol. 11, no. 7, pp. 67–70, 2020.

[15] J. N. He and W. Fan, "Design of unified identity authentication system with LDAP-based Web services," *Shaanxi Vocational Educ. Appl. Tech. Res.*, vol. 15, no. 2, pp. 12–18, 2020.

[16] R. Y. Zhou and G. M. Zhang, "Realization of keyboard hook algorithm with VB programming," *Mod. Comput.*, vol. 12, no. 6, pp. 198–200, 2009.

[17] Y. Z. Qiu, "Common security issues and repair recommendations for OAuth 2.0 authorization protocol," *Wireless Internet Technol.*, vol. 15, no. 7, pp. 45–48, 2018.

[18] OWASP. (2010). *Open Web Application Security Project (OWASP) Top Ten Project*. [Online]. Available: http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

[19] E. Y. Chen, Y. Pei, S. Chen, Y. Tian, R. Kotcher, and P. Tague, "OAuth demystified for mobile application developers," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2014, pp. 892–903.

[20] W. Li and C. J. Mitchell, "Analysing the security of Google's implementation of OpenID connect," in *Proc. 13th Int. Conf. Detection Intrusions Malware, Vulnerability Assessment*, 2016, pp. 357–376.

[21] C. Mainka, V. Mladenov, J. Schwenk, and T. Wich, "SoK: Single sign-on security—An evaluation of OpenID connect," in *Proc. IEEE Eur. Symp. Secur. Privacy*, Apr. 2017, pp. 189–202.

[22] B. Hamdane, A. Serhrouchni, A. Montfaucon, and S. Guemara, "Using the HMAC-based one-time password algorithm for TLS authentication," in *Proc. Conf. Netw. Inf. Syst. Secur. (SAR-SSI)*, La Rochelle, France, May 2011, pp. 1–8.

[23] D. Fett, R. Küsters, and G. Schmitz, "A comprehensive formal security analysis of OAuth 2.0," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2016, pp. 1204–1215.

[24] G. Sciarretta, R. Carbone, S. Ranise, and A. Armando, "Anatomy of the Facebook solution for mobile single sign-on: Security assessment and improvements," *Comput. Secur.*, vol. 71, pp. 71–86, Nov. 2017.

[25] M. Ghasemisharif, A. Ramesh, S. Checkoway, C. Kanich, and J. Polakis, "O single sign-off, where art thou? An empirical analysis of single sign-on account hijacking and session management on the Web," in *Proc. 27th USENIX Secur. Symp.*, 2018, pp. 1475–1492.

[26] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Trans. Inf. Theory*, vol. IT-29, no. 2, pp. 198–208, Mar. 1983.

[27] Y. N. Zhu, "Research on computational reliability of cryptographic protocol symbology analysis method," Ph.D. dissertation, School Cryptogr. Eng., PLA Inf. Eng. Univ., Zhengzhou, China, 2008.

[28] B. Blanchet, "A computationally sound mechanized prover for security protocols," in *Proc. IEEE Symp. Secur. Privacy*, May 2006, pp. 140–154.

[29] J. Chen and Q. J. Wang, "Input security protection based on attack and defense perspective," *Silicon Valley*, vol. 15, no. 6, pp. 42–44, 2013.

[30] J. M. Fu, P. W. Li, and J. W. Li, "Research on sensitive information input security technology," *Inf. Netw. Secur.*, vol. 3, no. 6, pp. 152–160, 2013.

[31] A. Barth, C. Jackson, and J. C. Mitchell, "Robust defenses for cross-site request forgery," in *Proc. 15th ACM Conf. Comput. Commun. Secur. (CCS)*, New York, NY, USA, 2008, pp. 75–88.

[32] J. G. Mohamed and J. Visumathi, "A predictive model of machine learning against phishing attacks and effective defense mechanisms," *Proc. Mater. Today*, 2020, doi: 10.1016/j.matpr.2020.09.612.

[33] E. Raja and R. Ravi, "A performance analysis of software defined network based prevention on phishing attack in cyberspace using a deep machine learning with CANTINA approach (DMLCA)," *Comput. Commun.*, vol. 153, no. 4, pp. 375–381, Mar. 2020.

[34] G. Varshney, M. Misra, and P. Atrey, "Secure authentication scheme to thwart RT MITM, CR MITM and malicious browser extension based phishing attacks," *J. Inf. Secur. Appl.*, vol. 42, pp. 1–17, Oct. 2018.

[35] S. Bojjagania, D. R. D. Brabinb, and P. V. V. Raob, "PhishPreventer: A secure authentication protocol for prevention of phishing attacks in mobile environment with formal verification," in *Proc. 3rd Int. Conf. Comput. Netw. Commun. (CoCoNet)*, vol. 171, 2020, pp. 1110–1119.

[36] X. Tong, Z. Wang, Y. Du, and L. Yin, "Research on hybrid encryption technology based on software security," *Comput. Eng.*, vol. 30, no. 23, pp. 98–100, 2004.

[37] D. Kreutz, O. Malichevskyy, E. Feitosa, H. Cunha, R. da Rosa Righi, and D. D. J. de Macedo, "A cyber-resilient architecture for critical security services," *J. Netw. Comput. Appl.*, vol. 63, pp. 173–189, Mar. 2016, doi: 10.1016/j.jnca.2015.09.014.

[38] S. Qiu, G. Xu, H. Ahmad, and L. Wang, "A robust mutual authentication scheme based on elliptic curve cryptography for telecare medical information systems," *IEEE Access*, vol. 6, pp. 7452–7463, 2018, doi: 10.1109/ACCESS.2017.2780124.

[39] S. Qiu, D. Wang, G. Xu, and S. Kumari, "Practical and provably secure three-factor authentication protocol based on extended chaotic-maps for mobile lightweight devices," *IEEE Trans. Depend. Sec. Comput.*, early access, Sep. 8, 2020, doi: 10.1109/TDSC.2020.3022797.

[40] M. Nabil, M. H. A. Azeem, and M. H. Megahed, "Design and simulation of new one time pad (OTP) stream cipher encryption algorithm," *J. Adv. Res. Comput. Appl.*, vol. 10, no. 1, pp. 16–23, 2018.

[41] K. Chaturvedi, A. Matheus, S. H. Nguyen, and T. H. Kolbe, "Securing spatial data infrastructures for distributed smart city applications and services," *Future Gener. Comput. Syst.*, vol. 101, pp. 723–736, Dec. 2019.

[42] S.-T. Sun, K. Hawkey, and K. Beznosov, "Systematically breaking and fixing OpenID security: Formal analysis, semi-automated empirical evaluation, and practical countermeasures," *Comput. Secur.*, vol. 31, no. 4, pp. 465–483, Jun. 2012.

[43] P. Hu, R. Yang, Y. Li, and W. C. Lau, "Application impersonation: Problems of OAuth and API design in online social networks," in *Proc. 2nd ACM Conf. Online Social Netw. (COSN)*, 2014, pp. 271–278.

[44] F. Yang and S. Manoharan, "A security analysis of the OAuth protocol," in *Proc. IEEE Pacific Rim Conf. Commun., Comput. Signal Process.*, Aug. 2013, pp. 271–276.

[45] C. Bansal, K. Bhargavan, A. Delignat-Lavaud, and S. Maffeis, "Discovering concrete attacks on website authorization by formal analysis," *J. Comput. Secur.*, vol. 22, no. 4, pp. 601–657, 2014.

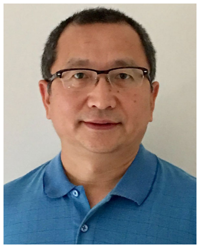
[46] R. Yang, G. Li, W. C. Lau, K. Zhang, and P. Hu, "Model-based security testing: An empirical study on OAuth 2.0 implementations," in *Proc. 11th ACM Asia Conf. Comput. Commun. Secur.*, May 2016, pp. 651–662.

[47] E. Birrell and F. B. Schneider, "Federated identity management systems: A privacy-based characterization," *IEEE Security Privacy*, vol. 11, no. 5, pp. 36–48, Sep. 2013.

[48] J. Werner and C. M. Westphall, "A model for identity management with privacy in the cloud," in *Proc. Symp. Comput. Commun.*, Jun. 2016, pp. 463–468.

[49] J. Zhuo, X. Li, J. Li, and J. Huai, "Attack classification and security assessment of security protocols," *Comput. Res. Develop.*, vol. 7, no. 42, pp. 1100–1107, 2005.

LIANGSHUN WU received the B.S. degree from Central South University, China, in 2014, and the M.S. degree from Wuhan University, China, in 2017, where he is currently pursuing the Ph.D. degree in software engineering and the M.S. degree in information and communication engineering with the Xi'an University of Posts and Telecommunications. His research interests include cryptography, network security, and embedded systems.



H. J. CAI received the B.S. degree in space physics from the University of Science and Technology of China, in 1984, the M.S. degree in space physics from the Center for Space Science and Applied Research, CAS, China, in 1989, and the Ph.D. degree in space physics from the University of Alaska Fairbanks, Fairbanks, AK, USA, in 1995. From 1986 to 1992, he was a Research Assistant with the Center for Space Science and Applied Research, CAS. From 1992 to 1996, he was a Research Assistant with the Geophysical Institute, University of Alaska Fairbanks. From 1996 to 1999, he held a postdoctoral position with the Department of Physics and Astrophysics, University of Iowa, Iowa City, IA, USA. From 1999 to 2005, he was with eNet Trade, Independent Realty Capital Corporation, and Bestprice Group Inc., USA. Since 2005, he has been a Professor with the International School of Software and the School of Computer Science, Wuhan University, China. He is currently a Full Professor and a Ph.D. Advisor with the School of Computer Science, Wuhan University, the Executive Director of the Zall Research Institute of Smart Commerce, a member of Ethics Committee of the Chinese Association for Artificial Intelligence (CAAI), an Expert Committee Member of China AI and Big Data Committee of 100, a Corresponding Member of China Computer Federation Technical Committee on Blockchain (CCF TCBC), and a Vice-President of the China Communications Industry Association Professional Committee of Blockchain (CCIAPCB).

HAN LI received the B.S. degree in software engineering from the Anhui University of Technology, China, in 2017, and the M.S. degree in control science and engineering from the National University of Defense Technology, China, in 2020. His current interest includes cross-media retrieval in the field of computer vision and data privacy computing.