SSUKey: A CPU-Based Solution Protecting Private Keys on Untrusted OS

Huorong Li[1,2,3], Wuqiong Pan[1,2](✉), Jingqiang Lin[1,2], Wangzhao Cheng[1,2,3], and Bingyu Li[1,2,3]

[1] Data Assurance and Communication Security Research Center, Beijing, China
[2] State Key Laboratory of Information Security, Institute of Information Engineering, CAS, Beijing, China
panwuqiong@iie.ac.cn
[3] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China

Abstract. With more and more websites adopt private keys to authenticate users or sign digital payments in e-commerce, various solutions have been proposed to secure private keys – some of them employ extra specific hardware devices while most of them adopt security features provided by general OS. However, users are reluctant to extra devices and general OS is too complicated to protect itself, let alone the private keys on it. This paper proposes a software solution, SSUKey, adopting CPU security features to protect private keys against the vulnerabilities of OS. Firstly, threshold cryptography (TC) is employed to partition the private key into two shares and two Intel SGX enclaves on local client and remote server are used to secure the key shares respectively. Secondly, the two enclaves are carefully designed and configured to mitigate the vulnerabilities of Intel SGX, including side channel and rollback. Thirdly, an overall central private key management is designed to help users globally monitor the usage of private keys and detect abnormal behaviors. Finally, we implement SSUKey as a cryptography provider, apply it to file encryption and Transport Layer Security (TLS) download, and evaluate their performance. The experiment results show that the performance decline due to SSUKey is acceptable.

Keywords: Trusted Execution Environment (TEE) · Intel SGX
Trusted computing · Threshold cryptography · Key protection

1 Introduction

Digital signature is widely used in authentication, digital payment and online banking. According to Stratistics MRC, the Global Digital Signature Market is accounted for $662.4 million in 2016 [1]. Digital signature is based on asymmetric cryptography which has a pair of public key and private key. The private key represents the identity of an entity and is used to create digital signatures while the public key is known to public and is used to verify the digital signatures. The private key should be kept secret.

At present the most effective way to protect private keys is using specific hardware devices, which have their own processors and storage isolated from host PC. This is adopted by Facebook, Google, GitHub, and Dropbox [2]. But users are reluctant to use specified hardware devices because they are inconvenient to carry and easy to lose. According to SafeNet Inc., the use of hardware-based authentication dropped from 60% in 2013 to 41% in 2014; conversely, the use of software-based authentication rose from 27% in 2013 to 40% in 2014 [3].

The security of software-based methods protecting private key relies on the security of privileged code, such as OS kernel and VMM (Virtual Machine Monitor). For example, [4,5] use hypervisor to provide isolation environment to protect sensitive data. However, privileged code had been found many serious vulnerabilities, for example, CVE-2015-2291, CVE-2017-0077, CVE-2016-0180 and CVE-2017-8468 for Windows kernel, CVE-2017-13715, CVE-2017-12146, CVE-2017-10663 and CVE-2016-10150 for Linux kernel, CVE-2009-1542, CVE-2016-7457 and CVE-2017-10918 for VMM, CVE-2016-8103, CVE-2016-5729 and CVE-2006-6730 for SMM, and may still have vulnerabilities.[1]

Intel Software Guard Extensions (SGX) [6–9] enables execution of security-critical application code, called enclaves, in isolation from the untrusted system software. It also provides enclaves processor-specific keys, such as the sealing key or the attestation key, which can be accessed by the enclaves. SGX is considered as a remarkable way to protect private keys when first proposed [7]. However, SGX has been found several vulnerabilities, such as cache-based side channel attack [10,11], page-based side channel attack [12], and rollback attack [9]. Although Intel has recently added support for monotonic counters (SGX counters) [13] that an enclave developer may use for rollback attack protection, this mechanism is likely vulnerable to bus tapping and flash mirroring attacks [14] since the non-volatile memory used to store the counters resides outside the processor package.

We propose a software solution, SSUKey, adopting Intel SGX to protect private keys against the vulnerabilities of OS. Especially, SSUKey employs ECC-based threshold cryptography (ECC-TC) to mitigate the vulnerabilities of SGX and enhance the security of SGX. Each private key is partitioned into two shares using ECC-TC, and the two key shares are protected using two Intel SGX enclaves on local client and remote server respectively. Since the two enclaves are carefully designed and the remote server can be carefully configured and well protected by additional mechanisms, such as advanced firewall, it is very difficult for an attacker to successfully perform side channel and rollback attacks on both the local client and the remote server enclaves. If the attacker only compromises one of them, threshold cryptography (TC) ensures that the attacker knows nothing about the private key. SSUKey also provides a central private key management. A user may use the same private key on different websites for con-

[1] All these vulnerabilities are published in Nation Vulnerability Database (NVD, https://nvd.nist.gov/).

venience. When the private key is compromised, our SSUKey can directly revoke the private key by the remote server immediately without having to inform all the websites respectively. The overall central private key management also help users globally monitor the usage of private keys and detect abnormal behaviors.

Windows CNG (Cryptography API: Next Generation), proposed by Microsoft, sets a standard interface for both cryptography provider and application. All Windows built-in applications (e.g., TLS, certificate tools, IE, Edge, IIS, etc.) use CNG to protect cryptographic keys and Microsoft recommends all Windows applications should use CNG to protect cryptographic keys. We implement our SSUKey complying with CNG, supporting SM2 (ECC algorithm) [15], SM3 (hash algorithm) [16], SM4 (symmetric algorithm) [17], PRF (pseudo random function) [18], and NIST hash-based DRBG (random number generation algorithm) [19]. As a proof-of-concept, we evaluate the single-thread performance of SSUKey on Intel NUC6 with i3-6100U CPU, which is designed low power (15 W). We first evaluate SSUKey by testing cryptographic operations. Compared to the software solution without any protection, the performance of verifying signatures, symmetric operations and hash operations almost does not change, while that of signing signatures declines from 481 Operations per second (Ops) to 110 Ops. Second, we evaluate SSUKey by testing it in real applications, file encryption and TLS download. Compared to the one without any protection, the performance of file encryption declines less than 3%, while that of TLS download (with 4 KB message) declines less than 1%. As the result, the performance decline due to SSUKey is acceptable.

In summary, we claim following main **contributions:**

- We propose and implement SSUKey, a CPU-based software solution protecting private keys against the vulnerabilities of OS, VMM, SMM, etc.
- Our SSUKey can mitigate the vulnerabilities of SGX, including side channel and rollback attacks, and add a useful function, an overall central private key management.
- We implement our SSUKey on Windows CNG, apply it to file encryption and TLS download, and evaluate the performance.

Intel CPU supports SGX starting with the Skylake microarchitecture, so our SSUKey can work on any new CPUs afterwards.

2 Assumptions

We consider a powerful adversary who controls all software except SSUKey on the target platform, including the OS. The adversary's aim is to compromise private keys. The adversary can block, delay, replay, read and modify all messages sent by SSUKey. Especially, the adversary can revert the sealed secrets in file system to previous state, i.e., rollback attack. The adversary can learn some information about the private keys by performing side channel attack [10,11].

The adversary cannot break through CPU and compromise SGX enclaves from inside. Specially the adversary cannot read or modify the enclave runtime memory and has no access to processor-specific keys, such as the sealing key or the attestation key. We also assume that it is very difficult to perform side channel or rollback attacks on both local client and remote server successfully, since the remote server is carefully configured and well protected by additional mechanisms, such as advanced firewall.

SSUKey ensures that the integrity, confidentiality and freshness of private keys. SSUKey does not aim to provide availability since the adversary controls the OS and denial-of-service (DoS) is always possible. SSUKey authenticates a user using the password entered by the user. But SSUKey does not protect the path between the input device like keyboard and the enclave. This function can be provided by SGXIO [20] which employs hypervisor to enhance the security of I/O path. Our SSUKey is compatible with SGXIO.

3 SSUKey Design

3.1 Architecture

Figure 1 shows our system overview. Our system consists of a remote server and some users' local platforms. Each local platform may run multiple user applications that host local client enclaves (CEs) which have an access to the user's cryptographic keys. The remote server runs a service that hosts a remote server enclave (SE) which assists CE to perform cryptographic operations cooperatively. The remote server is carefully configured and well protected by additional mechanisms, such as advanced firewall, intrusion detection system, and latest malware detection or anti-virus software. Both CE and SE run a share of ECC-TC algorithms and hold a share of corresponding cryptographic keys respectively.

Figure 2 gives the key components of our SSUKey architecture. Authentication and session management modules authenticate CE/SE to its counterpart and establish a trusted channel between CE and SE; sign/decrypt modules operate the cooperative ECC-TC algorithm shares and key management modules manage the key shares on CE/SE; authentication module authenticates user to CE; policy engine module checks while activity monitor module monitors the operation requests from CE overall; persistent storage stores the sealed key shares.

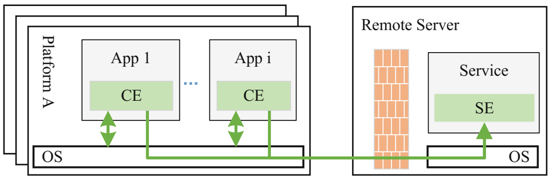


Fig. 1. System overview

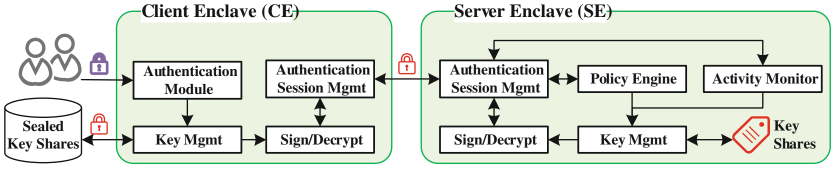


Fig. 2. SSUKey architecture

3.2 Trusted Channel Between CE and SE Establishment

A simple but effective protocol is proposed to establish a trusted channel between CE and SE with CE and SE authenticating each other. This protocol takes a few one-off steps to setup two asymmetric authentication key pairs. The one-off procedure proceeds as follows:

(1) During the first execution, SE generates an asymmetric authentication key pair SK_{SE}/PK_{SE}, and exports the public key. The public key is hard-coded to CE implementation.
(2) When a CE connects to SE for the first time, it generates an asymmetric authentication key pair SK_{CE}/PK_{CE}. The CE generates a SGX remote attestation report on the hash value of PK_{CE}. The report also includes the code measurement of the CE. After that, the CE encrypts PK_{CE} with SE's public key PK_{SE} and sends the ciphertext and the report to SE.
(3) Upon receiving the ciphertext and the report, SE verifies the report through attestation verification server typically provided by Intel, extracts the CE's public key PK_{CE}, and verifies whether the hash value of PK_{CE} matches that in the report.

On success, SE obtains the CE's public key (the CE has SE's public key hard-coded in its implementation). Specifically, SE is bounded with PK_{SE} and the CE is bounded with PK_{CE} since the key pairs can only be accessed from CE or SE respectively.

When a CE wants to connect to SE for the first time of current execution, the CE and SE use the raw public keys PK_{CE}/PK_{SE}, following the procedure specified in *RFC 7250: Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)* and TLS 1.3 [21], to establish a session key and use the established session key to protect all the subsequent messages between the CE and SE.

3.3 Key Setup

SSUKey adopts a 2-out-of-2 sharing scheme TC, Mediated SM2 [15,22], which partitions a private key into two shares. CE holds one share (denoted as d_{CE}) of the private key while SE holds the other share (denoted as d_{SE}). TC ensures that knowledge of one of the two key shares cannot not be used to derive the private key.

After successfully establishing a trusted channel between a CE and SE, the CE and SE setup a new key pair cooperatively follow the procedure as follows:

(1) The CE sends a *setup* opcode to SE.
(2) Upon receiving the opcode, SE generates a key share d_{SE}, allocates a *unique key identifier* ID and initializes *key status* STS for the key share. The key status can be one of three possible values, *valid*, *suspended*, and *revoked*, indicating the private key is available, suspended, and revoked respectively. A *suspended* key is not available until it is resumed while a *revoked* key is permanently not available. The key status is initialized *valid* when the key is created. After that, SE computes the public key share P_{SE} of d_{SE}.
(3) SE sends the ID and P_{SE} to the CE.
(4) Upon receiving the ID and P_{SE}, the CE generates the other key share d_{CE}, computes the public key P using P_{SE} and d_{CE}, and exports the public key.

The CE now has the public key P and a share of the private key d_{CE} while SE has the other share d_{SE} of the private key as well as the key identifier ID and the key status STS. The CE seals all the secrets (d_{CE}, P, and ID) two-fold – firstly, the secrets are sealed using user specific secret, such as password, and secondly, they are sealed using the CE's sealing key, which is derived from the code measurement of CE and the processor-specific secrets. The purpose of user specific secret is to authenticate the untrusted application that employs the CE and the application will be rejected to access the secrets if it fails to authenticate itself. After two-fold sealing the secrets, the CE saves the sealed secrets to persistent storage. The secrets on SE (ID, STS and d_{SE}) are saved in memory. When SE wants to store the secrets to non-volatile memory, SE seals them first and keeps the state of them in memory. The sealed secrets are protected from rollback attack as long as SE does not shut down since SE can maintain the state of the secrets itself. We explains more about rollback protection in Sect. 4.

3.4 Signature and Decryption

When a CE is starting, it firstly executes following steps in sequence. First, it loads the two-fold sealed secrets from persistent storage and extracts the key share d_{CE} and key identifier ID. Second, it establishes a trusted channel with SE. Once the trusted channel is established, the CE can trigger a signature or decryption operation by sending SE an opcode, *decrypt* or *sign*, as well as the message to be signed or decrypted. Upon receiving the opcode, SE signs or decrypts the message for the CE. The procedure proceeds as follows:

(1) The CE sends SE the key identifier ID, a *sign* or *decrypt* opcode, and the message.
(2) Upon receiving the ID and the opcode, SE checks the key policy, i.e., key status STS associated with the ID. If the key is available, SE signs or decrypts the message using d_{SE}.

(3) SE sends the result to the CE.
(4) Upon receiving the result, the CE continues signing or decrypting the result from SE using d_{CE} and obtains the final result, i.e., a signature or cleartext.

3.5 Key Management

SSUKey manages all private keys on SE. It globally monitors and analyzes the usage of private keys. For example, how frequent a private key is used, where a private key request is from, what a private key is typically used for (signing or decrypting), etc. Based on the private keys usage patterns, SE can detect abnormal behaviors. For example, the requested private key is far more frequent used in a short period. For another example, a signing operation requests a decrypting private key. If such abnormal behaviors are detected, SE suspends the associated private keys by setting key status to *suspended*. The private key owner will be informed next time when the CE requests to the suspended private key. This makes the owner aware of potential private key abusing. It is the owner's responsibility to confirm whether the suspended private key is still sound secure.

The private key owner can trigger a *resume* operation to resume a suspended private key, a *revoke* operation to revoke a suspended or unneeded private key, or a *sync* operation to synchronize current status of a private key. The procedure proceeds much like requesting a signature or decryption operation, except that it sends a key management opcode (*resume*, *revoke*, or *sync*) to SE. Upon receiving the opcode, SE updates the key status STS.

4 Security Analysis

The theoretical security of SSUKey is based on the security of ECC-TC [15,22] and the enclaves provided by Intel SGX. The adversary is allowed to attack from the very beginning of the private key being setup. The adversary has to compromise both two key shares on CE and SE separately to recover the private key. In this section, we mainly illustrate that the adversary cannot successfully compromise a private key by performing the most promising attacks on SSUKey, including tampering system memory, eavesdropping channels between CE and SE, performing side channel and rollback attacks on CE and SE.

Tamper-Proofing. The adversary cannot modify the code of CE and SE without being detected since any modification to the code will change the measurement of the code. A modified CE or SE will have a different code measurement comparing with the original one. CPU judges it as a different enclave. Thus, the modified CE or SE cannot access the secrets kept by the original one. The adversary cannot read or modify the runtime memory of CE and SE since the memory resides within the isolation region provided by SGX.

During the establishment of a session key between CE and SE, SE can authenticate the CE by verifying the signature of the session key signed by the CE's private key and the CE can confirm that only SE can decrypt the session key.

The adversary cannot masquerade as either CE or SE to establish a channel with the counterpart. Thus, a man-in-the-middle attack on SSUKey is not applicable. In addition, session hijack attack is disabled since the session key is immediately adopted once SE successfully authenticates the CE. The subsequent communication messages are transferred within the established trusted channel between CE and SE and the adversary cannot replay, read and modify the message.

Side Channel Protection. Intel SGX enclaves are vulnerable to side channel attack, for example, cached-based side channel attack [10,11] and page-based side channel attack [12]. CE and SE of our SSUKey are also threated by such attacks. A successful side channel attack on SSUKey has to extract the key shares from both the CE and SE. This is very difficult and almost impossible since the remote server that hosts SE is carefully configured and well protected by additional mechanisms. Compared with SE, the CE is more likely being attacked. If the key share d_{CE} on the CE is compromised due to side channel attack, TC ensures that the compromise of d_{CE} cannot be used to derive the private key.

SE verifies the CE's identity during the establishment of the trusted channel between the CE and SE. Thus, even though the adversary has compromised the d_{CE} on the CE, it cannot masquerade as the CE to request SE to help sign or decrypt a message. This makes SSUKey tolerant to the compromise of the key share d_{CE} on the CE.

Rollback Protection. Intel SGX enclaves are also vulnerable to rollback attack [9]. The adversary can exploit this vulnerability to break the freshness of the private key. A successful rollback attack on SSUKey has to revert the state of the two key shares from both the CE and SE. On the one hand, SE can be kept online almost all the time by a lot of ways (e.g., [23]), and the adversary cannot perform a rollback attack on SE successfully as long as SE does not shut down since SE can maintain the state of the secrets itself. On the other hand, when occasionally being restarted, for example, due to service update, SE can protect the secrets from rollback attack using SGX counters [13] or other useful solutions such as ROTE [24].

5 Evaluation

In this section, we describe our performance evaluation. We implemented our system consisting of two enclave libraries for CE and SE respectively, a remote service, and applications. The cryptographic library supports SM2 (Mediated SM2), SM3, SM4, KDF, and NIST hash-based DRBG. The enclave library for CE is implemented as a cryptography provider complying with Windows CNG. The internal distributed architecture of SSUKey is transparent to the applications. Both the applications and the remote service are running separately within a single thread atop Windows 10 on Intel NUC6 with i3-6100U CPU, which is designed low power (15 W). The applications connect to the remote service via local network (ping about 1.1 ms).

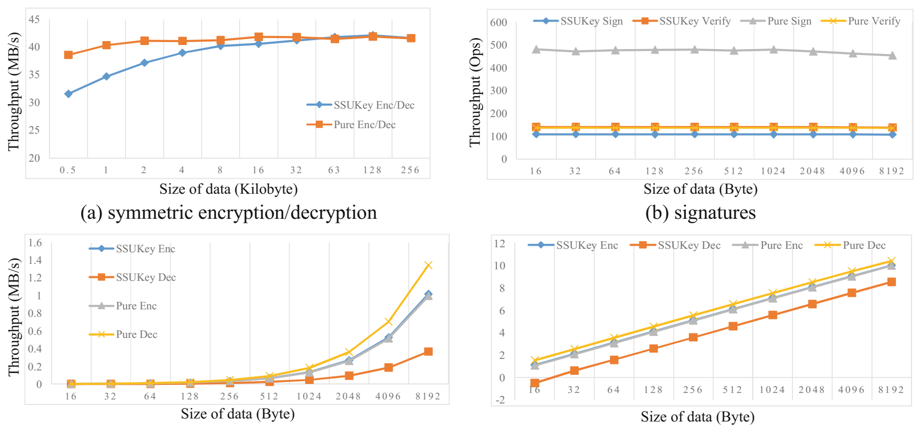
5.1 Cryptographic Operations Throughputs

The main performance metrics to measure are the throughputs of cryptographic operations including encryption, decryption, and signatures. A hash operation is included in a SM2 signature operation so we do not measure it alone.

We implemented several test cases for the cryptographic operations. We tested the test cases using (1) pure software implementation with neither enclaves nor Mediated SM2 and (2) our SSUKey implementation. For the symmetric encryption test case, it encrypted/decrypted the data repeatedly (data size varied from 0.5 KB to 256 KB). For the asymmetric encryption test case, it encrypted/decrypted the data repeatedly (data size varied from 16 B to 8 KB). For the signature test case, it signed/verified the data repeatedly (the data size varies from 16 B to 8 KB). We used data processed per second (MB/s) and operations per second (Ops) as the measuring unit to measure the throughputs of encryption and signatures respectively.

Figure 3a shows the performance of symmetric encryption/decryption. The throughputs are almost the same when data size is greater than 4 KB, while they are about 5–18% lower in SSUKey than that in the pure one when the data size is less than or equal to 4 KB. This is due to that data of small size weakens the throughput rate of a single operation and amplifies the influence from the overhead of enclave context switching.

The throughputs of asymmetric encryption and signature verification are almost the same. But the throughputs of asymmetric decryption and signature signing in SSUKey are about a quarter of that in the pure one, as shown in Figs. 3b and c (right figure is the logarithms of the throughputs to show the difference more clearly). This is as expected, since SSUKey adopts ECC-TC (i.e., Mediated SM2), and the procedure of using public keys (i.e., encrypting or verifying) is identical in both SSUKey and the pure one, while the procedure of using



(a) symmetric encryption/decryption

(b) signatures

(c) asymmetric encryption/decryption

Fig. 3. The performance of cryptographic operations

private keys (i.e., signing or decrypting) in SSUKey adopts a sharing scheme but the pure one does not. The sharing scheme brings SSUKey much more time-consuming operations, e.g., point multiplication and large integer multiplication. Additionally, an asymmetric operation is generally considered much more time-consuming than that a symmetric operation and thus the overhead of enclave context switching is weakened in an asymmetric operation.

5.2 Applications Throughputs

Additionally, we evaluated our SSUKey in real-world scenarios, file encryption and TLS download. We implemented a file encryption application, a TLS server, and a TLS client that connected to and downloaded data (4 KB) from the TLS sever repeatedly. The TLS server acted as a download center and waited for the TLS client to connect.

We tested the file encryption application and the TLS client using (1) pure software implementation with neither enclaves nor Mediated SM2 and (2) our SSUKey implementation. The file encryption application encrypted/decrypted files repeatedly (file size varied from 0.5 MB to 256 MB). The TLS client connected to and downloaded data (4 KB) from the TLS sever repeatedly. We used the number of successful downloads per second (Ops) as the measuring unit to measure the throughput of TLS download.

Figure 4 shows the performance of file encryption. For file size greater than 8 MB, the throughput of file encryption in SSUKey is almost the same with that in the pure one, while it is 7–19% decline in SSUKey than that in the pure one when file size is less than or equal to 8 MB. As for TLS download, the performance is 72.04 Ops in SSUKey while 72.15 Ops in the pure one. The performance decline is less than 1%.

The results illustrates that our SSUKey imposes a moderate overhead to file encryption and has little influence on TLS download. We conclude that our SSUKey is acceptable.

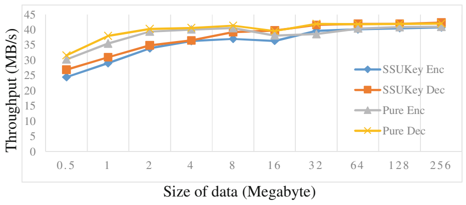


Fig. 4. The performance of file encryption

6 Related Work

TrustZone. ARM TrustZone (TZ) combines secure execution with trusted path support. It provides one secure world isolated against a normal world. The secure world operates a whole trusted stack, including security kernel, device drivers and applications. TZ allows device interrupts being directly routed into the secure world and thus supports generic trusted paths [25]. However, TZ does not distinguish between different secure application processes in hardware. It requires a security kernel for secure process isolation, management, attestation and similar. The prototype of SSUKey is promising to be migrated to TZ. Compared with SGX, TZ is more competent to offer generic trusted I/O path.

Rollback Protection. SGX counters [13] is a moderate and handy solution. It employs non-volatile memory to store the counter which is likely vulnerable to bus tapping and flash mirroring attacks [14]. It is not secure enough in our threat model since non-volatile memory resides outside the processor package. ROTE [24] is a more secure and promising solution than SGX counters. It adopts a distributed architecture and synchronizes the status of counters between distributed systems. This makes ROTE provide rollback protection counter as long as the status of the counter is kept on one or more systems. SSUKey is compatible with ROTE but we do not implement ROTE in this work.

7 Conclusion

In this paper, we have presented SSUKey, a CPU-based solution protecting private keys. Our main idea is to adopt Intel SGX to resist the vulnerabilities of privileged code, including OS kernel, and employ ECC-TC to mitigate the vulnerabilities of SGX, including side channel and rollback. We consider a powerful adversary that controls the OS and has even compromised one share of the private key. We provide a central key management function to help users globally monitor the usage of private keys and detect the abnormal behaviors, minimizing the risk of private key abusing. Our experiments demonstrate that our SSUKey is acceptable with a moderate performance decline when compared with the one without protection from SGX and TC.

Acknowledgments. We thank the anonymous reviewers for their helpful feedback. The work was partially supported by the National Basic Research 973 Program of China (No. 2014CB340603) and the National Natural Science Foundation of China (No. 61772518).

References

1. Stratistics MRC: Digital Signature - Global Market Outlook (2016–2022). http://www.strategymrc.com/report/digital-signature-market. Accessed Sept 2017
2. Services that Integrate with the YubiKey. https://www.yubico.com/solutions/#FIDO-U2F. Accessed Sept 2017
3. SafeNet Inc.: 2014 Authentication Survey Executive Summary. https://safenet.gemalto.com/news/2014/authentication-survey-2014-reveals-more-enterprises-adopting-multi-factor-authentication/. Accessed Sept 2017

4. Hofmann, O., et al.: InkTag: secure applications on an untrusted operating system, vol. 41, pp. 265–278. ACM (2013)
5. McCune, J., et al.: TrustVisor: efficient TCB reduction and attestation. In: 2010 IEEE Symposium on Security and Privacy (SP), pp. 143–158. IEEE (2010)
6. Anati, I., Gueron, S., Johnson, S., Scarlata, V.: Innovative technology for CPU based attestation and sealing. In: Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy, vol. 13 (2013)
7. Hoekstra, M., et al.: Using innovative instructions to create trustworthy software solutions, p. 11 (2013)
8. McKeen, F., et al.: Innovative instructions and software model for isolated execution. In: HASP@ ISCA, p. 10 (2013)
9. Costan, V., Devadas, S.: Intel SGX explained. IACR Cryptology ePrint Archive, 2016:86 (2016)
10. Schwarz, M., et al.: Malware guard extension: using SGX to conceal cache attacks. arXiv preprint arXiv:1702.08719 (2017)
11. Brasser, F., et al.: Software grand exposure: SGX cache attacks are practical. arXiv preprint arXiv:1702.07521 (2017)
12. Xu, Y., Cui, W., Peinado, M.: Controlled-channel attacks: deterministic side channels for untrusted operating systems. In: 2015 IEEE Symposium on Security and Privacy (SP), pp. 640–656. IEEE (2015)
13. Intel: SGX documentation: SGX create monotonic counter. https://software.intel.com/en-us/node/709160. Accessed Sept 2017
14. Skorobogatov, S.: The bumpy road towards iPhone 5c NAND mirroring. arXiv preprint arXiv:1609.04327 (2016)
15. Shen, S. (ed.): SM2 Digital Signature Algorithm (draft 02) (2014). https://tools.ietf.org/html/draft-shen-sm2-ecdsa-02
16. Shen, S. (ed.): SM3 Hash function (draft 01) (2014). https://tools.ietf.org/html/draft-shen-sm3-hash-01
17. Tse, R.: The SM4 Block Cipher Algorithm and Its Modes of Operations (draft 01) (2014). https://tools.ietf.org/html/draft-ribose-cfrg-sm4-01
18. Dierks, T.: RFC 5246: the transport layer security (TLS) protocol. The Internet Engineering Task Force (2008)
19. Barker, E., Kelsey, J.: Recommendation of random number generation using deterministic random bit generators. NIST SP800-90A, June 2015
20. Weiser, S., Werner, M.: SGXIO: Generic Trusted I/O Path for Intel SGX. arXiv preprint arXiv:1701.01061 (2017)
21. Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.3 (draft 21) (2017). https://tools.ietf.org/pdf/draft-ietf-tls-tls13-21.pdf
22. Lin, J., et al.: Signing and decrypting method and system applied to cloud computing and based on SM2 algorithm (2014). http://www.soopat.com/Patent/201410437599. CN Patent CN104243456A
23. Li, D., Morton, P., Li, T., Cole, B.: Cisco hot standby router protocol (HSRP) (1998)
24. Matetic, S., et al.: ROTE: rollback protection for trusted execution. IACR Cryptology ePrint Archive 2017:48 (2017)
25. Li, W., et al.: Building trusted path on untrusted device drivers for mobile devices. In: APSys 2014. ACM (2014)