Recent Advances in the Web PKI and the Technical Challenges in SCMS

Yunkun Wu[1], Xiaokun Zhang[2(✉)], Yajun Teng[3], Zhenya Liu[4], Liang Huang[1], Jingqiang Lin[4], and Xuhua Bao[1]

[1] Legendsec Information Technology (Beijing) Inc., Beijing, China
{wuyunkun,huangliang,baoxuhua}@qianxin.com
[2] Academy of Opto-Electronics, Chinese Academy of Sciences, Beijing, China
xkzhang@aoe.ac.cn
[3] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
tengyajun@iie.ac.cn
[4] School of Cyber Security, University of Science and Technology of China, Hefei, China
hhhhlzy@mail.ustc.edu.cn, linjq@ustc.edu.cn

Abstract. The Web PKI plays a more and more important role in network security, as nowadays TLS and HTTPS are being widely adopted. The most significant recent advances in the Web PKI include certificate transparency and push-based revocation, which improve the trustworthiness and performance of TLS and HTTPS, respectively. Meanwhile, SCMS is a specialized PKI system designed for V2V communications. In this paper, we analyze the design principles of certificate transparency and push-based revocation, study the similar requirements in V2V communications, and then summarize the technical challenges to integrate certificate transparency and push-based certificate revocation into SCMS. From the experiences and lessons in the Web PKI, we do believe that the current designs of SCMS are still not completely ready to be deployed in the real world.

Keywords: Public key infrastructure (PKI) · Transport layer security (TLS) · Certificate · Security credential management system (SCMS) · Trust management · Vehicle-to-Vehicle (V2V) communication

1 Introduction

Public key infrastructures (PKIs) [11,28] provide various security services such as confidentiality, authentication, data integrity, and non-repudiation, through certificates signed by a trusted certification authority (CA). The Web PKI (or sometimes called the TLS/HTTPS PKI) is the PKI system that is implemented and deployed for web security, especially for TLS [13] and HTTPS [46]. In the Web PKI, a list of accredited root CAs are trusted by the mainstream operating

B. Chen and X. Huang (Eds.): AC3 2021, LNICST 386, pp. 155–167, 2021.
https://doi.org/10.1007/978-3-030-80851-8_11

systems (OSs) and browsers (e.g., Microsoft Windows, Apple macOS, Google Android, Mozilla Firefox, etc.) [3,40,43]. These root CAs and their subordinate CAs are responsible for signing TLS server certificates binding a domain name (e.g., www.facebook.com or www.gmail.com) to a key pair held by the website. Then, a browser will establish secure TLS sessions with the website, after verifying the TLS server certificate binding the to-be-visited domain.

As TLS and HTTPS are being widely adopted after the structure of PKI systems has been proposed for more than thirty years, several technical advances in the Web PKI are proposed and implemented recently. First of all, *certificate transparency* is proposed and deployed [34,50], to detect fraudulent TLS server certificates and improve the trustworthiness and accountability of CAs. Several security incidents indicate that even accredited famous CA systems may be compromised, deceived or even compelled to issue fraudulent TLS server certificates [10,16,19,25,41,48,56,59,60], which bind a domain name to a key pair held by man-in-the-middle (MitM) or impersonation attackers, instead of the legitimate website. Certificate transparency depends on redundant public log servers to record all CA-signed TLS server certificates, and browsers accept a TLS server certificate only if it is recorded in multiple independent publicly-visible logs [2,21,42]. Thus, a fraudulent certificate will be found by the victim website soon, which acts as a monitor or visits the third-party monitors [9,18,22,45] to search for certificates of interest in the public log servers.

Meanwhile, another significant technical advance of the Web PKI is *push-based certificate revocation*. In the traditional design of the Web PKI, a browser checks the revocation status of a TLS server certificate, through certificate revocation list (CRL) [11] or online certificate status protocol (OCSP) [44]. That is, the browser has to download the CRL file or acquire the OCSP response by itself, through another connection to the PKI system, in addition to the TCP connections to the visited website. The addition connection delays remarkably impact the performance of TLS and HTTPS [31,32,39,49]. On the contrary, push-based revocation requires (a) the certificate holder (i.e., the visited webserver in TLS and HTTPS) to actively send the OCSP message in TLS negotiations, e.g., OCSP stapling [15] and OCSP must-staple [24], or (b) the PKI system to proactively push all or most certificate revocation status data to browsers through the manufacturers, e.g., CRLSet [52], OneCRL [20] and CRLite [29,33]. Push-based revocation eliminates the addition connections for revocation status data.

On the other hand, the security credential management system (SCMS) [58], is a specialized PKI system introduced for vehicle-to-vehicle (V2V) communications. Compared with traditional PKI systems, *short-lived pseudonym certificate*, implicit certificates [7,8], butterfly key expansion, and *linkage-based revocation* are designed in SCMS, to balance security, privacy, and efficiency in the V2V communications. Before SCMS is deployed widely in the real world (except some pilot projects [53]), the experiences and lessons in the large-scale Web PKI are very useful for us. However, we find that *fraudulent certificates* and *certificate revocation* are not carefully considered in the designs of SCMS. Therefore, in this paper, we analyze the possibilities to integrate certificate transparency and

push-based certificate revocation into SCMS, and then presents the technical challenges in the integration. These studies help to improve SCMS and also other PKI systems for V2V communications [6,58].

The remainder of this paper is organized as follows. Section 2 describes certificate transparency and push-based revocation, and Sect. 3 in details discusses the technical challenges to integrate certificate transparency and push-based certificate revocation into SCMS. Finally, we conclude this paper in Sect. 4.

2 Recent Advances in the Web PKI

In the traditional Web PKI, a CA is responsible for signing TLS server certificates. In TLS negotiations, a browser does not accept a TLS server certificates until it verifies the CA's signature on the certificate. Meanwhile, the browser also needs to check the revocation status of the TLS server certificate. That is, the browser by itself downloads the CRL file based on the CRL distribution points extension in the certificate, or acquires the OCSP response based on the AIA OCSP extension [11,44]. The CA is usually also responsible for signing CRL files and OCSP responses, or sometimes these functions are implemented by independent CRL issuers or OCSP servers [11,44].

As TLS and HTTPS are widely adopted in the Internet, the original designs of the Web PKI are improved recently. The recent advances include certificate transparency and push-based certificate revocation, which are not included in the original designs of the Web PKI [11,28].

2.1 Certificate Transparency

In the original design of PKIs, a CA is fully trusted to be responsible for signing a certificate only after carefully communicating with the applicant. But security incidents indicate that CA systems may be compromised, deceived or compelled to issue fraudulent certificates [10,16,19,25,41,48,56,59,60]. Therefore, in addition to CAs, browsers and websites, certificate transparency introduces the following PKI components [34].

Log Server. A log server maintains publicly-visible append-only logs that record certificates. It accepts certificates from CAs. All certificate records in a log are organized as a Merkle hash tree, and the root node is periodically signed by the log server, called the signed tree heads (STHs). There are more than one hundred log servers in the Internet in 2020 [23], and a TLS server certificate is recorded redundantly in multiple logs [55].

Monitor. Monitors regularly watch for suspicious certificates in the public logs. A monitor regularly fetches certificates from these logs, decodes the certificates, and searches for the certificates of interest among them. A website may assume the monitor role by itself [34] to search for the certificates binding its domain name, and there are also third-party monitors [9,18,22,45] which process the records in public logs to provide convenient certificate search services for users.

Auditor. Auditors ensure the correct behaviors of log servers. An auditor archives the STHs of a log. Then, by comparing any two STHs and requiring the consistency proof from the log server, an auditor checks whether a log is strictly append-only, i.e., any version of the log is a sub-tree of any later version. It also checks that each recorded certificate corresponds to an entry in the publicly-visible log [34], by verifying the audit path (i.e., the shortest list of additional nodes in the Merkle tree to compute the STH). An auditor may be a stand-alone service, or an internal component of a browser or a monitor.

After signing a TLS server certificate, the CA submits it to some log server. The log server responds with a signed certificate timestamp (SCT), which is a promise to append the certificate to the public log within the maximal merge delay (MMD). Then, SCTs are sent along with the certificate in TLS negotiations. In addition to the CA's signature and the certificate revocation status data, a browser also verifies the log servers' signatures in these SCTs. This implies that a browser preinstalls the public keys of trusted log servers [2,23] as well as the self-signed certificates of trusted root CAs.

It is worth noting that during TLS negotiations the browser does *not* establish addition connections to log servers, to check whether the TLS server certificate has been recorded in the public logs or not. The browser only verifies the log servers' signatures in the SCTs, and in the future some auditors will checks whether each SCT (i.e., a recorded certificate) corresponds to an entry in the log or not.

The above operations of certificate transparency ensure that any certificate accepted by browsers will be visible to the website (or certificate subject), with the help of monitors [35,37]. Thus, the website that is aware of all legitimate certificates issued with its authorization, will find the fraudulent ones among them if any.

2.2 Push-Based Certificate Revocation

In the original designs of the Web PKI, certificate revocation status data are obtained by the certificate verifiers (i.e., browsers) [11,28,44]. On the contrary, push-based certificate revocation eliminates the addition connections to obtain the certificate revocation status data. Typical approaches are listed as below.

OCSP Stapling. OCSP stapling [15] is a TLS extension that enables the website to send an OCSP response in TLS negotiations, and this OCSP message proves the validity (or unrevokedness) of its TLS server certificate. Note that the website acquires the OCSP response from the CA (or OCSP server) in advance. Thus, the browsers do not need the addition connections to check the certificate revocation status. Moreover, in order to defend against the downgrade attacks that exploit revoked TLS server certificates but do not send OCSP stapling extensions in TLS negotiations, the certificate extension of OCSP must-staple is defined [24]. A TLS server certificate with an OCSP must-staple extension, must be sent along with OCSP stapling messages in TLS negotiations; otherwise, it will be immediately rejected by the browsers. OCSP must-staple prevents the

downgrade attacks, even when a browser does not download the CRL files or access the OCSP server by itself.

Proactive CRL. Browser manufacturers propose to *proactively* (or periodically) push CRL files to browsers [20,29,52]. Thus, the browsers utilize these CRL files to check the certificate revocation status in TLS negotiations. The dilemma of certificate coverage vs. CRL size exists in the solutions of proactive CRL: in order to cover all (or even the most) of the TLS server certificates, the proactively-pushed CRL file will be greater than 100 MB [33]; on the other hand, small-sized CRL will frequently fail in the checking of certificate revocation status. So CRLite [33] and Let's Revoke [47] recently propose more efficient data structures to encode the revocation status data into small-sized messages, to periodically push the revocation status of *all* TLS server certificates to browsers.

3 The Technical Challenges in SCMS

This section firstly discusses the V2V PKI system. Then, we present the structure of SCMS, and the technical challenges to integrate certificate transparency and push-based certificate revocation into SCMS.

SCMS is designed for V2V communications, as well as other V2V PKI solutions [6,53], especially for the continuous broadcast of basic safety messages (BSMs) by the on-board equipment (OBE) device in each vehicle. It is estimated that BSMs will prevent most of the roadway crashes through active safety applications [54].

3.1 The Expected Properties of V2V PKI Systems

We briefly list the expected properties of V2V PKI systems. These properties shall be considered in certificate signing and certificate verification.

- **Trustworthiness.** As a security infrastructure for cyber-physical systems, the V2V PKI services shall be highly trustworthy. Any defects in this system will cause physical damages and bring direct profits to the attackers, so it needs to be well-protected.
- **User Privacy.** The most primary privacy concern is to protect users against vehicle tracking. That is, it shall be very difficult for the eavesdroppers in physically distant locations to tell whether two BSMs are sent by the same vehicle or not. Frequent change of different pseudonym certificates (e.g., every 10 min) is the common design.
- **High-Volume.** A V2V PKI system shall be able to support hundreds of millions of vehicles, and this number is increasing. Note that the number of pseudonym certificates may be several thousands times (i.e., hundreds of billions), when user privacy is considered and the frequent change of certificates is adopted.

– **Efficiency.** Wireless BSMs are broadcast at least every 100 ms among vehicles, and processed by the embedded OBE devices. Active safety applications usually do not tolerate any latency greater than 0.2 ms. So the security-related operations must be efficient, especially certificate verification in the process of received BSMs.

3.2 SCMS

We list the special features of the certificate services in SCMS as below, while we skip the steps of common certificate services (e.g., CA hierarchy and enrollment certificate) and some special steps but unrelated to fraudulent certificate and certificate revocation (e.g., butterfly key expansion and implicit certificate). More details of SCMS can be found in [58].

In the SCMS PKI, each certified OBE device holds a long-lived enrollment certificate after the device bootstrap. Then, this enrollment certificate is used to apply for multiple short-lived pseudonym certificates, which are then used to sign/verify BSMs. Each OBE device holds 20–40 simultaneously-valid pseudonym certificates for every week, and it applies for pseudonym certificates for signing the BSMs of 1–3 years (i.e., about 1040–6240 pseudonym certificates) in a batch [58]. In order to protect user privacy, any pair of these pseudonym certificates cannot be linked, unless two SCMS internal components collude or these certificates are revoked.

SCMS depends on CRL to revoke certificates, and CRL files are broadcast by road side equipments or satellite radio systems to all OBE devices in the roadway. SCMS does not consider OCSP, because it is impractical for a vehicle to acquire OCSP responses frequently, either the BSM sender or the verifier. That is, CRL files are periodically pushed to all OBE devices, so that an OBE device will locally maintain the identifiers of all revoked certificates. Moreover, linkage-based revocation is designed to reduce the size of CRL files in SCMS, and an entry in linkage-based CRL represents all unexpired but revoked pseudonym certificates for which a certain OBE device applies in a batch [27,58]. With linkage-based revocation, a revocation identifier (or CRL entry) in the CRL file represents a batch of pseudonym certificates: if the relationship of the identifier (i.e., serial number) of a pseudonym certificate and the revocation identifier satisfies the specified rules, the pseudonym certificate is considered as revoked.

3.3 Technical Challenge #1: Certificate Transparency vs. Pseudonym Certificate

Certificate transparency is proposed against compromised CAs of the Web PKI [10,16,19,25,41,56,59,60]. Although SCMS has not been widely deployed in the real world yet, it is reasonable to assume that some CAs in SCMS might be compromised or deceived to sign fraudulent pseudonym certificates for software vulnerabilities and cyber attacks frequently happen. Then, a fraudulent pseudonym certificate enables an attacker (but not the certified OBE

device embedded in some vehicle) to arbitrarily broadcast BSMs, to impersonate a vehicle in the road. So certificate transparency and/or other countermeasures [5,17,30,51,57] are also necessary to enhance the trustworthiness of SCMS in the future.

Firstly, as mentioned in Sect. 2.1, certificate transparency depends on the victim website by itself to search for certificates binding its domain name, to detect fraudulent certificates in the public logs. However, in SCMS a *pseudonym* certificate does not bind any meaningful identity or identifier. So no one is able to search for these certificates to detect fraudulent ones, even when all SCMS pseudonym certificates are publicly recorded in log servers. Certificate transparency involves the certificate search based on the identifiers of certificate holders (e.g., domain names) [34], but pseudonym certificates have to mask these identifiers. Therefore, the security principle of certificate transparency does not work well for the pseudonym certificates in SCMS.

Next, we further analyze other solutions which are proposed to tame the absolute authority of CAs [4,38] against possible fraudulent certificates. The existing schemes include:

- **Public key pinning** [17]**.** A browser locally pins the certificates (or public keys) of a TLS server for the visited domain, after a successful TLS negotiation. The pinned public key will be compared with the received certificates in the future TLS negotiations, and then any mismatching is detected immediately by the browser.
- **Restricted scopes of services** [30,48]**.** A CA of the Web PKI is restricted to serve only some scopes of domains, and the restriction rules are enforced by browsers when verifying the TLS server certificates. A certificate violating the rules will be rejected by browsers.
- **Multi-path verification** [1,57]**.** On receiving a server certificate in TLS negotiations, a browser compares it with other copies obtained through different network paths (e.g., an extra Tor circuit). The certificate is accepted, only if they are identical.
- **Subject-controlled policies** [26,51]**.** The certificate subject (or website) specifies its own certificate policies (e.g., a list of authorized CAs), and these policies are published in a publicly-visible means. Any TLS server certificate violating these policies (e.g., a certificate signed by an unauthorized CA) is considered as invalid or fraudulent.
- **Multi-authority certification** [5,51]**.** A TLS server certificate is certified and signed redundantly by multiple independent CAs, and the browser verifies all CAs' signatures. Only when all signatures are valid, the certificate is considered as valid.

Let's analyze the scenarios when these schemes are integrated into SCMS for V2V communications. Public key pinning requires a certificate verifier (i.e., another vehicle or a road side equipment receiving BSMs) to maintain the pubic keys of communication peers, but it is really unsuitable for SCMS because every OBE device holds 20–40 simultaneously-valid pseudonym certificates and each

pseudonym certificate expires in days [58]. That is, a pinning mismatching happens frequently but normally in the V2V communications. Restricted scopes of services only mitigate the attack impact of fraudulent certificates but cannot completely prevent or detect the attacks, so it is not enough for cyber-physical systems. Multi-path verification introduces addition connections and greatly degrades the performance, and it brings false alarms when the website (or vehicle) holds multiple valid certificates simultaneously.

Similar to certificate transparency, subject-controlled policies do not work for pseudonym certificates, for the subject certificate policies shall be published by the long-lived enrollment certificate. Thus, in order to validate pseudonym certificates with such a policy will inevitably break the user privacy. Finally, multi-authority certification will remarkably increase the delay of certificate verification due to the expensive public-key cryptographic computation of multiple signature verifications.

Therefore, it is rather difficult but imperative to design different solutions to prevent or detect (possible) fraudulent certificates in SCMS, which work for pseudonym certificates and delay-sensitive communications. Combining (the security principles of) these existing schemes into a specialized solution may work for V2V communications, such as Elaphurus [38] for the Web PKI. However, since the V2V communications do not tolerate high delays, a solution focuses on the steps of certificate signing but not certificate verification may be more practical.

3.4 Technical Challenge #2: Push-Based Certificate Revocation vs. the Great Volume of Pseudonym Certificates

Push-based certificate revocation is proposed to improve the efficiency of certificate verification. If certificate revocation is possible (e.g., the OBE device is compromised), such efficiency improvements of revocation status checking are always required in SCMS. SCMS is designed for V2V communications, which are very delay-sensitive. The push model has actually been adopted in SCMS already, and all CRL files are broadcast to OBE devices in the vehicles [58]. Therefore, because the number of certificates in SCMS is much greater than that in the Web PKI, the dilemma of certificate coverage vs. CRL size also exists in SCMS (or will become even worse), and the efficient data structures to encode certificate revocation status data such as CRLite [33] and Let's Revoke [47] are more imperative.

Due to the great volume of pseudonym certificates, the size of CRL files may bring very heavy burdens when SCMS is deployed in the real world. There are hundreds of millions of vehicles, and hundreds of billions of pseudonym certificates will be signed in SCMS [58]. These numbers are much greater than those of the Web PKI, where there are only 24–32 millions unrevoked TLS server certificates and about 12 millions revoked ones [33]. This implies that the size of CRL files in SCMS will be at least 10 times that of the Web PKI according to the number of users, or the size will be even 10,000 times that of the Web PKI according to the number of pseudonym certificates. For example, Apple signs a

CRL file over 76 MB [39] in the Web PKI, and it is very expensive or even impossible for an embedded OBE device to receive and store such CRL files (or raw certificate revocation status data) through wireless communications. Even if the certificate revocation rates are roughly equal, the CRL file may be about 1 GB in the case of linkage-based revocation (i.e., the CRL size is estimated according to the number of users, but not the number of certificates); while if the certificate revocation rate increases due to some security incidents (e.g., OpenSSL Heartbleed resulted in an irregular 40-fold increase of revocation rate [14,61]), the CRL file will be probably expanded to even several GB. In fact, as typical embedded systems, the OBE devices might be exposed to more physical attack surfaces than TLS webservers with professional administrators, and then the certificate revocation rate of SCMS could become greater.

We next study the recently-proposed efficient designs of data structures for push-based certificate revocation (i.e., CRLite [33] and Let's Revoke [47]). CRLite utilizes the Bloom-filter cascade to encode the identities of both all revoked certificates and all unrevoked ones. This introduces at least two challenges as follows: (a) CRLite requires certificate transparency to fetch all unrevoked certificates from the public logs; otherwise, some valid certificates may be falsely checked as revoked due to the inherent false positive of Bloom filters; and (b) linkage-based revocation of SCMS cannot work compatibly with CRLite, because it requires the *explicit* identifier of a batch of pseudonym certificates in the revocation status data [58] while CRLite cannot extract or recover the identifiers of revoked certificates after they have been encoded into the Bloom-filter cascade. Meanwhile, Let's Revoke depends on a special rule to generate deterministic certificate serial numbers (i.e., a sequence of incremental integers as certificate identifiers, so that only one bit is enough to indicate the revocation status in CRL), but this rule conflicts with linkage-based revocation in SCMS where a certificate serial number is computed by XORing two *random* linkage values, generated by two independent linkage authorities [58]. If the CA assigns another field as the certificate identifier for Let's Revoke, linkage-based revocation will not work and then the size of revocation status data will become 1,000 times.

Besides, the optimization of proactive CRL distributions utilizing the locality of reference [12,36] requires a locally-centralized gateway to cache certificate revocation status data, through either CRL or OCSP. However, such a gateway does not exist in the dynamic wireless V2V communications.

In summary, neither CRLite nor Let's Revoke works compatibly in SCMS. Meanwhile, although linkage-based revocation has been designed in SCMS to reduce the size of CRL, this size reduction is still not enough based on the experiences and lessons in the Web PKI. So it still needs a more efficient data structure to push the revocation status data in SCMS. In the future, we plan to integrate linkage-based revocation and Let's Revoke (or other efficient data structures). For example, Let's Revoke is utilized to revoke a *batch* of certificate signing which results in about 1040–6240 pseudonym certificates for one vehicle, and linkage-based revocation helps to find all identifiers of these pseudonym certificates.

4 Conclusions

Unexpected technical challenges usually appear and then advances are finished, as a security design is deployed in the real world. Although the structure of PKI systems has been discussed, analyzed, implemented and evaluated for more than thirty years, recent advances still happen in the Web PKI as TLS and HTTPS are being widely adopted in the Internet. In particular, certificate transparency and push-based revocation are the most significant recent advances in the Web PKI, to (a) improve the trustworthiness against fraudulent TLS server certificates and (b) eliminate the addition connections for certificate revocation status data, respectively.

SCMS is a specialized PKI system for V2V communications. The number of certificates in SCMS will become much greater than that in the Web PKI, and the V2V communications for BSM broadcast are very delay-sensitive. Therefore, we do believe that the problems solved by certificate transparency and push-based revocation in the Web PKI will appear again when SCMS is deployed in the real world. In this paper, we study these technical challenges in SCMS and find that the existing approaches in the Web PKI do not work compatibly for SCMS. These discussions will be useful references to improve the designs of SCMS in the future.

Acknowledgement. This work was partially supported by National Key RD Plan of China (Grant No. 2020YFB1005803) and National Natural Science Foundation of China (Grant No. 61772518).

References

1. Alicherry, M., Keromytis, A.: DoubleCheck: multi-path verification against man-in-the-middle attacks. In: 14th IEEE Symposium on Computers and Communications (ISCC) (2009)
2. Apple Inc.: Apple's certificate transparency policy (2019). https://support.apple.com/en-us/HT205280
3. Apple Inc.: Lists of available trusted root certificates in macOS (2020). https://support.apple.com/en-us/HT202858
4. Bao, X., Zhang, X., Lin, J., Chu, D., Wang, Q., Li, F.: Towards the trust-enhancements of single sign-on services. In: 3rd IEEE Conference on Dependable and Secure Computing (DSC) (2019)
5. Basin, D., Cremers, C., Kim, H., Perrig, A., Sasse, R., Szalachowski, P.: ARPKI: attack resilient public-key infrastructure. In: 21st ACM Conference on Computer and Communications Security (CCS) (2014)
6. Bibmeyer, N., Stubing, H., Schoch, E., Gotz, S., Stolz, J., Lonc, B.: A generic public key infrastructure for securing car-to-X communication. In: 18th World Congress on Intelligent Transport Systems (ITS) (2011)
7. Brown, D.R.L., Gallant, R., Vanstone, S.A.: Provably secure implicit certificate schemes. In: Syverson, P. (ed.) FC 2001. LNCS, vol. 2339, pp. 156–165. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-46088-8_15
8. Campagna, M.: SEC 4: Elliptic curve Qu-Vanstone implicit certificate scheme (ECQV). Technical report, Certicom Research (2013)

9. Comodo CA Limited: crt.sh: Certificate search (2020). https://crt.sh
10. Comodo Group Inc.: Comodo report of incident (2011). https://www.comodo.com/Comodo-Fraud-Incident-2011-03-23.html
11. Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., Polk, W.: IETF RFC 5280 - Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile (2008)
12. Dickinson, L., Smith, T., Seamons, K.: Leveraging locality of reference for certificate revocation. In: 35th Annual Computer Security Applications Conference (ACSAC) (2019)
13. Dierks, T., Rescorla, E.: IETF RFC 5246 - The transport layer security (TLS) protocol (2008)
14. Durumeric, Z., et al.: The matter of Heartbleed. In: 14th Internet Measurement Conference (IMC) (2014)
15. Eastlake, D.: IETF RFC 6066 - Transport layer security (TLS) extensions: extension definitions (2011)
16. Eckersley, P.: A Syrian man-in-the-middle attack against Facebook (2011). https://www.eff.org/deeplinks/2011/05/syrian-man-middle-against-facebook
17. Evans, C., Palmer, C., Sleevi, R.: IETF RFC 7469 - Public key pinning extension for HTTP (2015)
18. Facebook Inc.: Facebook: Certificate transparency monitoring (2020). https://developers.facebook.com/tools/ct/search/
19. GlobalSign: Security incident report (2011). https://www.globalsign.com/resources/globalsign-security-incident-report.pdf
20. Goodwin, M.: Revoking intermediate certificates: introducing OneCRL (2015). https://blog.mozilla.org/security/2015/03/03/revoking-intermediate
21. Google Inc.: Certificate transparency in Chrome (2018). https://github.com/chromium/ct-policy/blob/master/ct_policy.md
22. Google Inc.: Google: HTTPS encryption on the web (2018). https://transparencyreport.google.com/https/certificates
23. Google Inc.: Known logs (2018). http://www.certificate-transparency.org/known-logs
24. Hallam-Baker, P.: IETF RFC 7633 - X.509v3 transport layer security (TLS) feature extension (2015)
25. Adkins, H.: An update on attempted man-in-the-middle attacks (2011). https://security.googleblog.com/2011/08/update-on-attempted-man-in-middle.html
26. Hoffman, P., Schlyter, J.: IETF RFC 6698 - The DNS-based authentication of named entities (DANE) transport layer security (TLS) protocol: TLSA (2012)
27. IEEE Vehicular Technology Society: IEEE 1609.2: Standard for wireless access in vehicular environments (2016)
28. ITU-T: X.509: Information technology - Open systems interconnection - The directory: Authentication framework (1997)
29. Jones, J.C.: CRLite: Speeding up secure browsing (2020). https://blog.mozilla.org/security/2020/01/21/crlite-part-3-speeding-up-secure-browsing/
30. Kasten, J., Wustrow, E., Halderman, J.A.: CAge: taming certificate authorities by inferring restricted scopes. In: Sadeghi, A.-R. (ed.) FC 2013. LNCS, vol. 7859, pp. 329–337. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39884-1_28
31. Langley, A.: No, don't enable revocation checking (2014). https://www.imperialviolet.org/2014/04/19/revchecking.html
32. Langley, A.: Revocation still doesn't work (2014). https://www.imperialviolet.org/2014/04/29/revocationagain.html

33. Larisch, J., Choffnes, D., Levin, D., Maggs, B., Mislove, A., Wilson, C.: CRLite: a scalable system for pushing all TLS revocations to all browsers. In: 38th IEEE Symposium on Security and Privacy (S&P) (2017)
34. Laurie, B., Langley, A., Kasper, E.: IETF RFC 6962 - Certificate transparency (2014)
35. Li, B., et al.: Certificate transparency in the wild: exploring the reliability of monitors. In: 26th ACM Conference on Computer and Communications Security (CCS) (2019)
36. Li, B., Lin, J., Wang, Q., Wang, Z., Jing, J.: Locally-centralized certificate validation and its application in desktop virtualization systems. IEEE Trans. Inf. Forensics Secur. (TIFS) **16**, 1380–1395 (2021)
37. Li, B., Chu, D., Lin, J., Cai, Q., Wang, C., Meng, L.: The weakest link of certificate transparency: exploring the TLS/HTTPS configurations of third-party monitors. In: 18th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom) (2019)
38. Li, B., Wang, W., Meng, L., Lin, J., Liu, X., Wang, C.: Elaphurus: ensemble defense against fraudulent certificates in TLS. In: Liu, Z., Yung, M. (eds.) Inscrypt 2019. LNCS, vol. 12020, pp. 246–259. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-42921-8_14
39. Liu, Y., et al.: An end-to-end measurement of certificate revocation in the web's PKI. In: 15th Internet Measurement Conference (IMC) (2015)
40. Microsoft Corporation: Microsoft trusted root certificate program: participants (2020). https://gallery.technet.microsoft.com/Trusted-Root-Program-d17011b8
41. Morton, B.: More Google fraudulent certificates (2014). https://www.entrust.com/google-fraudulent-certificates/
42. Mozilla: Mozilla CT policy (2019). https://groups.google.com/a/chromium.org/forum/m/#!topic/ct-policy/Xx1bv8r33ZE
43. Mozilla: Mozilla included CA certificate list (2020). https://wiki.mozilla.org/CA/Included_Certificates
44. Myers, M., Ankney, R., Malpani, A., Galperin, S., Adams, C.: IETF RFC 2560 - X.509 Internet public key infrastructure online certificate status protocol - OCSP (1999)
45. Opsmate Inc.: SSLMate: Cert spotter (2020). https://sslmate.com/certspotter/
46. Rescorla, E.: IETF RFC 2818 - HTTP over TLS (2000)
47. Smith, T., Dickenson, L., Seamons, K.: Let's revoke: scalable global certificate revocation. In: 27th ISOC Network and Distributed System Security Symposium (NDSS) (2020)
48. Soghoian, C., Stamm, S.: Certified lies: detecting and defeating government interception attacks against SSL (short paper). In: Danezis, G. (ed.) FC 2011. LNCS, vol. 7035, pp. 250–259. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-27576-0_20
49. Stark, E., Huang, L.S., Israni, D., Jackson, C., Boneh, D.: The case for prefetching and prevalidating TLS server certificates. In: 19th ISOC Network and Distributed System Security Symposium (NDSS) (2012)
50. Stark, E., et al.: Does certificate transparency break the web? Measuring adoption and error rate. In: 40th IEEE Symposium on Security and Privacy (S&P) (2019)
51. Szalachowski, P., Matsumoto, S., Perrig, A.: PoliCert: secure and flexible TLS certificate management. In: 21st ACM Conference on Computer and Communications Security (CCS) (2014)
52. The Chromium Projects: CRLSets (2015). https://dev.chromium.org/Home/chromium-security/crlsets

53. US Department of Transportation: Safety pilot model deployment (2012). http://safetypilot.umtri.umich.edu/
54. US Department of Transportation: Vehicle-to-vehicle (V2V) communications for safety (2020). https://one.nhtsa.gov/Research/Crash-Avoidance
55. VanderSloot, B., Amann, J., Bernhard, M., Durumeric, Z., Bailey, M., Halderman, A.: Towards a complete view of the certificate ecosystem. In: 16th Internet Measurement Conference (IMC) (2016)
56. VASCO Data Security International Inc.: DigiNotar reports security incident (2011). https://www.vasco.com/about-vasco/press/2011/news_diginotar_reports_security_incident.html
57. Wendlandt, D., Andersen, D., Perrig, A.: Perspectives: improving SSH-style host authentication with multi-path probing. In: USENIX Annual Technical Conference (ATC) (2008)
58. Whyte, W., Weimerskirch, A., Kumar, V., Hehn, T.: A security credential management system for V2V communications. In: 5th IEEE Vehicular Networking Conference (VNC) (2013)
59. Wikipedia: Flame (malware) (2017). https://en.wikipedia.org/wiki/Flame_(malware)
60. Wilson, K.: Distrusting new CNNIC certificates (2015). https://blog.mozilla.org/security/2015/04/02/distrusting-new-cnnic-certificates/
61. Zhang, L., et al.: Analysis of SSL certificate reissues and revocations in the wake of Heartbleed. In: 14th Internet Measurement Conference (IMC) (2014)

The First International Workshop on Security for Internet of Things (IOTS 2021)

A Novel Approach for Code Smells Detection Based on Deep Learning

Tao Lin[1(✉)], Xue Fu[2(✉)], Fu Chen[3(✉)], and Luqun Li[2(✉)]

[1] Amazon, Seattle, WA 98109, USA
paper@Ltao.org
[2] Shanghai Normal University, Shanghai, China
success@shnu.edu.cn
[3] Central University of Finance and Economics, Beijing, China
chenfu@cufe.edu.cn

Abstract. Compared to software bugs, code smells are more significant in software engineering research. It is not easy to detect code smells through traditional methods. In this work, we propose a novel code smells detection approach based on deep learning. The experiments show that our work achieves high scores in terms of F2 score.

Keywords: Code smells · Deep learning · Convolutional neural network

1 Introduction

Code smells are increasingly generated by modern agile software development. This is because code changes are much more frequent and occur on a daily basis for large software companies and dominant open-source communities.

Although there are many more test approaches to detect code smells, these methods have some defects. Due to the frequent changes, it is increasing probable to generate code smells overheads. Code smells, like software bugs, are a serious problem in modern software.

Nowadays, the code smells are being researched by many practitioners. Software developers are not aware of what is the code smell, although they are aware of software bugs, thanks integrated environment development kits that can provide many instant suggestions and notifications when there are bugs.

The question here is how to detect code smells effectively? And what is the motivation for detecting code smells? Although there are many more test approaches to detect code smells, these methods have some defects. There are mainly two categories of deep learning networks. One is Recurrent Neural Networks, and another is Convolutional networks.

Convolutional networks have already demonstrated its usage by leveraging hierarchy features. In this paper, we use fully convolutional networks for code smells detection based on semantic features. We will use fully convolutional networks for this work.

The original version of this chapter was revised: there is a typo in the chapter title: “leaning” has been corrected to: “learning”. The correction to this chapter is available at https://doi.org/10.1007/978-3-030-80851-8_16

B. Chen and X. Huang (Eds.): AC3 2021, LNICST 386, pp. 171–174, 2021.
https://doi.org/10.1007/978-3-030-80851-8_12

We will define the type of neural network we use, and explain how it is used to detect code smells. An advantage of using convolutional network is its ability to identify and use local correspondences.

In recognition and machine learning, convolutional networks are increasingly significant. Convolutional network presents the improvement on image recognition. An example is using convolutional network on local correspondence. In software engineering, we definitely can use these kinds of information for code smells detection.

To our knowledge, this is the first work to train a convolutional network for code smells recognition. The inference is much more improved through convolutional network.

This work is an extension of the authors previous work [8].

Unlike previous works that needs additional information for code smells detection, this work does not use any existing information for code smells detection. One of the major challenges in code smells detection is to find the relationship between code semantics and code location. There is a tradeoff between identifying the correct semantics compared to identifying the correct location of the smells.

Although there are several success stories from image recognition by using deep networks [1]. It is hard to transfer these approaches to software engineering, which is more deterministic. Fully convolutional network has been used for one-layered computation, and has a potential to be deployed to multi-layered environments.

2 Code Smells Detection Based on Convolutional Networks

We can define a multi-dimensional array to represent the convolutional network, h * w * d, where h and w are space dimensions, and d is the channel. The first layer is our source code inputs.

The second layer is the networks for sequence modeling. For example, the inputs are x_0, x_1, x_2, x_3, x_4,… x_n, and the outputs are y_0, y_1, y_2, y_3, y_4,… y_n. The second layer will be y'_0, y'_1, y'_2, y'_3, y'_4,… y'_n.

The outputs will be reshaped to a one-dimensional array, where size will be D *1024. This output array will be dilation blocks. For the encoder task, we should process noise. Each layer in the encoder is processed by normalization and liner analysis.

3 High Level Design

With recent development in software engineering, it is easy to find software bugs using several methods from compiling to running. Our work is based on the state-of-the-art deep learning methods for detection and recognition task.

Firstly, we transform the software source code to XML file, in order to be processed by deep learning models [2]. Then there are two steps: code segment proposal and classification. The code segment proposal leverages the heuristic search to generate following inputs. These segments are processed by CNN classifier. We try to avoid to use R-CNN, otherwise. One of the main reasons is that R-CNN uses selective search algorithm, which is time consuming.

We use the following equation for segments:

$$\text{Seg} = \left(\max rp^{2^{l}/D} + minrq\right) * (I/D)$$

Seg is the segments, r is the rule of limits, and p is process variable, q is the next graph inputs, I is interception, and D is next destination.

3.1 Experiments Results

We use an open-source database published by the authors' previous work [11].

The experiments results are shown as following table:

Table 1. Experiments results

	Precision	Recall	F-score	Kappa
Long method	0.528	0.674	0.754	0.635
Lazy class	0.624	0.678	0.613	0.632
Speculative generality	0.712	0.734	0.689	0.643
Refused bequest	0.698	0.701	0.711	0.677
Duplicated code	0.543	0.568	0.594	0.585
Contrived complexity	0.783	0.792	0.810	0.802
Shotgun surgery	0.597	0.596	0.501	0.601
Uncontrolled side effects	0.801	0.799	0.805	0.810

From Table 1, this work achieves high performance in terms of F2 score, especially for the category of uncontrolled side effects and contrived complexity.

4 Conclusion

In this work, we conducted a research for code smells detection based on deep learning.

Our solution uses convolutional neural network for training a model to detect several common code smells problems in software engineering. The solution achieves satisfied F2 score with the average above 0.75.

Acknowledgements. Part of this work is from the author's PhD study [1], before the author joining Amazon. Professor Fu Chen from Central University of Finance and Economics provided many constructive suggestions and perspectives for this work during author's PhD study. Professor Fu Chen and this work was supported in part by National Science Foundation of China under No. 61672104.

References

1. Lin, T.: A Data Triage Retrieval System for Cyber Security Operations Center. Pennsylvania State University Thesis (2018)
2. Lin, T.: A container - Destructor – Explorer paradigm to code smells detection. J. Chin. Comput. Syst. **37**(3), 17 (2016)
3. Lin, T., Fu, X.: Flame detection based on SIFT algorithm and one class classifier with undetermined environment. Comput. Sci. **42**(6), 6A (2015)
4. Lin, T., Zhong, C., Yen, J., Liu, P.: Retrieval of relevant historical data triage operations in security operation centers. In: Samarati, P., Ray, I., Ray, I. (eds.) From Database to Cyber Security. LNCS, vol. 11170. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-04834-1_12
5. Lin, T.: A novel image matching algorithm based on graph theory. Comput. Appl. Softw. **33**(12), 156 (2016)
6. Lin, T.: Graphic user interface testing based on petri net. Appl. Res. Comput. **33**(3), 27 (2016)
7. Lin, T.: A novel direct small world network model. J. Shanghai Norm. Univ. **45**(5), 23 (2016)
8. Lin, T., Gao, J., Fu, X., Lin, Y.: A novel bug report extraction approach. In: Wang, G., Zomaya, A., Perez, G.M., Li, K. (eds.) ICA3PP 2015. LNCS, vol. 9532, pp. 771–780. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-27161-3_70
9. Zhong, C., Lin, T., Liu, P., Yen, J., Chen, K.: A cyber security data triage operation retrieval system. Comput. Secur. **76**, 12–31 (2018)
10. Lin, T.: Deep learning for IoT. In: 39th IEEE – International Performance Computing and Communications Conference (2020)
11. Lin, T.: Security Operations Center Retrieval (2021). https://github.com/ltaocs/SecurityOperationsCenterRetrieval

A Thieves Identification Scheme for Prepaid Systems in Smart Grids

Zhiyuan Sui(✉), Hengyue Jia, Fu Chen, and Jianming Zhu

Information School, Central University of Finance and Economics, Beijing, China

Abstract. This paper proposes a privacy preserving thieves identification scheme for prepaid Smart Grid systems. Prepaid systems allow consumers to buy credentials from an operation center before consumption. According to the credentials, consumers can use the corresponding amount of electricity. Based on the dynamic k-times anonymous authentication protocol, the scheme can achieve thieves identification and privacy preservation at the same time without the involvement of a trusted authority in the system. Finally, we point out the path of our future work.

Keywords: Smart grid · Anonymity · Privacy · Security

1 Introduction

With the development of public awareness of environmental conservation, more and more renewable energy come to use. However, the renewable energy sources are so volatile that the power companies have to employ the Information and Communication Technologies (ICT) to balance the power production and consumption. The power company aggregates the power usage reports from consumers to calculate the amount of power consumption. The usage reporting device at each customer site is called as smart meter. The operation center and smart meters form the Smart Grid. The invention of Smart Grids is a great plus to the electricity industry. This mechanism can be exploited to improve energy efficiency and infrastructure reliability. However, trustless data, which does not represent the real consumption, may damage the electricity grid infrastructure [1]. At the same time, the billing information is generated by the usage data. Greedy thieves are not willing to send the trustworthy data and pay for their consumption. Hence, it is a serious challenge to resist fake data and detect power thieves in Smart Grids. On the other hand, privacy is another key requirement in Smart Grids. The usage data from the consumer should not result in finding consumers' usage information, which could lead to disclosure of the consumer's living habits or production outputs, and further causes personalized advertisements or intelligence leakage [2].

To solve the security and privacy preservation challenges, power request model was proposed for prepaid card system. The prepaid smart card system

B. Chen and X. Huang (Eds.): AC3 2021, LNICST 386, pp. 175–180, 2021.
https://doi.org/10.1007/978-3-030-80851-8_13

allows consumers buy credentials from the power company in advance. When the consumer needs power in the existing privacy preserving prepaid card system, he just sends the usage plan with his transformed credentials to the operation center. The operation center will send the power back according to the number of credentials [3]. However, credentials may be collision due to large data base. Moreover, the scheme [3] cannot identify thieves.

Taking the requirements of privacy preservation and thieves identification into account, in this paper, we construct a secure and privacy preserving thieves identification scheme in Smart Grids. There are two important virtues: (1) the scheme resists credential collisions; (2) energy thieves can be identified in the prepaid system.

The rest of this paper is organized as follows. Section 2 discusses related work in the prepaid system domain. Section 3 elaborates and explains necessary system models. In Sect. 4, we describe our proposed scheme. Finally, the paper is concluded in Sect. 5.

2 Related Work

To solve the security and privacy preservation challenges, power request models for smart grid systems have been proposed [3]. In such models, consumers receive tokens signed with blind signatures from the power provider and authenticate themselves using blinded tokens. The tokens represent a corresponding denomination of energy cost. Consequently, the power provider knows the total amount of customer energy costs but do not know the details of the usage information. However, this scheme cannot identify power thieves. Dimitriou et al. [4] added a proof to the blind signature. According to the proof, reused tokens from the same consumer can be identified. However, the tokens are generated by the consumers, and token collisions are inevitable. Zhao et al. [5] employed a fully homomorphic encryption algorithm to aggregate smart meters' usage data. Based on the homomorphism of the ciphers, consumers' billing can be calculated without knowing the plaintexts. However, current fully homomorphic encryption algorithms are less efficient. Xue et al. [6] improved the Paillier encryption algorithm. The exponent on the cipher is also additive homomorphism. Therefore, it can also achieve dynamic billing management with less computational cost. Li et al. [7] divided the usage data into multiple parts according to their denominations. In their study, each smart meter holds a corresponding number of key pairs with respect to the divided sets. Therefore, smart meters belonging to different sets should pay their corresponding billings.

Smart Grids cannot ensure improvement in infrastructure reliability without trustworthy information. At the same time, consumers do not wish their energy usage data to be exposed to the operation center. However, existing thieves identification approaches cannot make use of a trade-off between privacy and security. According to prepaid card systems, this paper proposes a privacy-preserving thieves identification scheme without any trusted authority. In this scheme, a thief can be traced if sending a used credential.

3 The Framework

In this section, the assumed network and security requirements in the scheme are described.

3.1 Network Model

Nowadays, in order to improve energy efficiency and infrastructure reliability, a prepaid system is proposed [3]. In the scheme, the system model mainly consists of two entities: the operation center (OC) and the smart meter (SM). The number of smart meters is large enough for each smart meter to cloak its usage behavior. As depicted in Fig. 1, firstly, SMs join the Smart Grid and obtain their commitments from the OC. Secondly, each SM buys credentials from the OC. Thirdly, a SM sends blinded commitments and credentials in an anonymized form when it needs to purchase electricity.

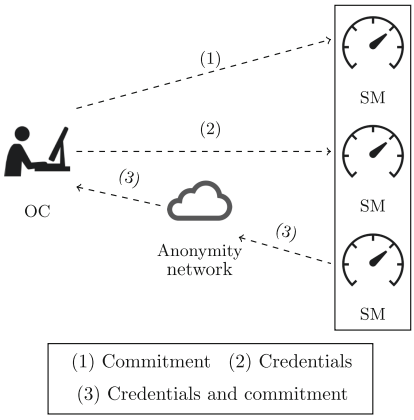

Fig. 1. Network model.

3.2 Security Requirements

The privacy-preserving thieves identification scheme is expected to exhibit the following properties:

1. **Privacy Preservation:** Curious eavesdroppers cannot obtain consumers' usage profiles from the usage reports from smart meters.
2. **Unlinkability:** The adversary cannot link different usage reports from the same smart meter.
3. **Authentication:** The operation center ensures that the usage reports are from legitimate smart meters in the Smart Grid.
4. **Traceability:** Energy thieves can be traced if they attempt to send used credentials.

4 Our Proposed Approach

In this section, a privacy-preserving thieves identification scheme is proposed based on the dynamic k-times anonymous authentication scheme.

4.1 Setup

In the privacy-preserving thieves identification scheme, a concrete region is managed by a single operation center. The operation center runs the setup algorithm on input of the security size κ to obtain a group public key and a secret key as follows:

1. Let p be a prime number of size κ, $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ be three cyclic groups of prime order p. Suppose P_1 and P_2 are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively, and e is a bilinear map. On input of κ, the bilinear pairing instance generator returns a tuple $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathbb{Z}_p^*, e, P_1, P_2)$.
2. Then, the operation center chooses collision-resistant hash functions $\mathcal{H}_{\mathbb{Z}_p^*} : \{0,1\}^* \to \mathbb{Z}_p^*$ and $\mathcal{H}_{\mathbb{G}^2} : \{0,1\}^* \to (\mathbb{G}_1, \mathbb{G}_1)$, two elements $V, Q \in \mathbb{G}_1$ and $\gamma \in \mathbb{Z}_p^*$, and computes $P_{\text{pub}} = \gamma P$.
3. Finally, the operation center retains its secret key γ, publishes its public key $(P_1, P_2, V, Q, P_{\text{pub}})$, and builds an empty authentication log **LOG** that records the used credentials.

4.2 Joining

The joining protocol is carried out between the operation center and a smart meter. Each household or company is equipped with a smart meter in the Smart Grid system. A secure public key signature scheme, including a signing algorithm **sig** and a verification algorithm **ver**, is selected for a smart meter. In this protocol, the smart meter must reveal its unique identity ID_j to the operation center as follows. Firstly, the smart meter generates a secure parameter $x_j \in \mathbb{Z}_p^*$, computes a request $C_j = x_j P_1$ and generates a signature $\sigma = \mathbf{sig}(C_j \| \text{ID}_j)$. The smart meter sends the signature σ with the request C_j and identity ID_j to the operation center. Upon the receipt, the operation center computes a hash value $f_j = \mathcal{H}_{\mathbb{Z}_p^*}(\text{ID}_j)$, grants the request $S_j = \frac{1}{(f_j+\gamma)}(C_j + Q)$ and replies S_j. Upon receipt, the smart meter confirms that the equation $e(S_j, f_j P_2 + P_{\text{pub}}) = e(x_j P_1 + Q, P_2)$ holds. Finally, the smart meter retains its secret key x_j and publishes its public key (C_j, S_j).

4.3 Power Purchasing

The power-purchasing protocol is carried out between the operation center and a smart meter. Using this protocol, the smart meters buy credentials from the operation center and are granted the right to obtain electricity.

Periodically, the operation center publishes the bound number $k > 0$ and calculates the set of public credentials $\{(t_i, \hat{t}_i) = \mathcal{H}_{\mathbb{G}^2}(i, n_T) \mid 1 \leq i \leq k\}$,

where n_T denotes the timestamp. The public credential $(t_i, \hat{t}_i)$ is called the ith credential base of the operation center. If q smart meters want to buy k units of electricity, represented by k credentials, the smart meters send their identities to the operation center. The operation center calculates $\{f_j = \mathcal{H}_{\mathbb{Z}_p^*}(\mathrm{ID}_j) \mid 1 \leq j \leq q\}$ and $W = \sum_{j=1}^{n}(\gamma + f_j)V$ and $V_j = \frac{1}{\gamma + f_j}W$ for the jth smart meter. The operation center publishes $\{(t_i, \hat{t}_i) \mid 1 \leq i \leq k\}$ and sends V_j to the smart meter ID_j. The smart meter operates a counter μ_j, which is initially set to zero.

4.4 Power Requesting

The power-requesting protocol is run by the operation center and a smart meter to prove knowledge of a smart meter's key x_j and an identity hash value f_j. The smart meter transforms its commitment and credentials, and sends them back to the operation center. Therefore, the operation center cannot locate the source of the public parameters even though the credentials generated by itself.

1. Firstly, the smart meter analyzes the usage data and estimates the amount of energy m. The smart meter increases the counter number by m.
2. If $\mu_j > k$, the smart meter will jump to the power-purchasing algorithm; otherwise, the smart meter sets $\mu_j = \mu_j + m$, chooses m credentials denoted as $(t_1, \hat{t}_1)$, ..., $(t_m, \hat{t}_m)$, and outputs m hash values $\{c_i = \mathcal{H}_{\mathbb{Z}_p^*}(t_i \| \hat{t}_i \| n_T) \mid 1 \leq i \leq m\}$, where n_T denotes the timestamp.
3. After that, the smart meter computes the credentials $\{(\Gamma_i = x_j t_i, \hat{\Gamma}_i = f_j t_i + c_i x_j \hat{t}_i) \mid 1 \leq i \leq m\}$, and generates a proof using the following non-interactive zero-knowledge proof:

$$\left\{ \begin{pmatrix} S_j \\ x_j \\ f_j \\ V_j \end{pmatrix} : \begin{array}{l} e(S_j, f_j P_2 + P_{\mathrm{pub}}) = e(x_j P_1 + Q, P_2) \\ e(V_j, f_j P_2 + P_{\mathrm{pub}}) = e(W, P_2) \\ (\Gamma_i = x_j t_i, \hat{\Gamma}_i = f_j t_i + x_j c_i \hat{t}_i) \end{array} \right\}$$

   The smart meter proves its credentials by sending the proof and transformed credential to the operation center. Please refer to [8] for the proof.
4. After receiving the proof and credential, the operation center checks the validity of the timestamp. If it is valid, the operation center compares the credential $(\Gamma_i, \hat{\Gamma}_i)$ with all corresponding credentials in **LOG**, and checks if it is different to the credentials in **LOG**. Finally, the operation center verifies that the proof does prove knowledge of its identity information.

4.5 Identification

If the operation center validates the signature σ and discovers that a received credential has already been used, it will run the identification algorithm. Suppose there are two credentials (Γ_i, n_T) and (Γ_i', n_T'), where $\Gamma_i = \Gamma_i'$ and $n_T \neq n_T'$. The operation center can confirm that a credential has been used more than once.

Then, the operation center computes $c = \mathcal{H}_{\mathbb{Z}_p^*}(t_i \| \hat{t}_i \| n_T)$, $c' = \mathcal{H}_{\mathbb{Z}_p^*}(t_i \| \hat{t}_i \| n'_T)$ and $\beta = \frac{c'\hat{\Gamma}_i - c\hat{\Gamma}'_i}{c-c'}$, searches for the ID_j which satisfies $\beta = \mathcal{H}_{\mathbb{Z}_p^*}(\mathrm{ID}_j)t_i$.

The public keys $\{(t_i, \hat{t}_i) \mid 1 \leq i \leq k\}$ are produced by a collision-resistant function, making the transformed credentials $\{(\Gamma_i, \hat{\Gamma}_i) \mid 1 \leq i \leq k\}$ also resistant to collisions. Therefore, malicious smart meters cannot deny their misbehavior.

5 Conclusion

Energy theft occurs frequently in Smart Grids due to its large amount of consumers. In order to defend against energy theft under prepaid smart grid systems, this paper proposes a scheme to achieve identification of thieves without a trusted party. Only an operation center and smart meters are required in the scheme. The operation center checks the smart meters' commitments and corresponding credentials. Therefore, the credentials are resistant to collisions. Great computational capacity is required for operation centers. For the future work, we will study possible ways and improve k-times anonymous authentication scheme to reduce the computational cost during power request procedures.

Acknowledgement. This work is supported by Funds of the National Natural Science Foundation of China (U201509214), (61672104), (62072487) and (61906220).

References

1. Gunduz, M.Z., Das, R.: Cyber-security on smart grid: threats and potential solutions. Comput. Netw. **169**(14), 107094 (2020)
2. Ferraga, M.A., Maglarasc, L.A., Janickec, H., Jiang, J.: A systematic review of data protection and privacy preservation schemes for smart grid communications. Sustain. Cities Soc. **38**, 806–835 (2018)
3. Chim, T.W., Yiu, S.M., Hui, L.C.K., Li, V.O.-K.: Privacy-preserving advance power reservation. IEEE Commun. Mag. **50**(8), 18–23 (2012)
4. Dimitriou, T., Karame, G.: Enabling anonymous authorization and rewarding in the smart grid. IEEE Trans. Dependable Secure Comput. **14**(5), 565–572 (2017)
5. Zhao, S., Li, F., Li, H., Lu, R., Ren, S.: Smart and practical privacy-preserving data aggregation for fog-based smart grids. IEEE Trans. Inf. Forensics Secur. **16**, 521–536 (2021)
6. Xue, K., et al.: PPSO: a privacy-preserving service outsourcing scheme for real-time pricing demand response in smart grid. IEEE Internet Thins J. **6**(2), 2486–2496 (2019)
7. Li, S., Zhang, X., Xue, K., Zhou, L., Yue, H.: Privacy-preserving prepayment based power request and trading in smart grid. Chin. Commun. **15**(4), 24–37 (2018)
8. Nguyen, L., Safavi-Naini, R.: Dynamic k-times anonymous authentication. In: Ioannidis, J., Keromytis, A., Yung, M. (eds.) ACNS 2005. LNCS, vol. 3531, pp. 318–333. Springer, Heidelberg (2005). https://doi.org/10.1007/11496137_22

Using Smart Contracts to Improve Searchable Symmetric Encryption

Yanbing Wu[1] and Keting Jia[2(✉)]

[1] Institute for Advanced Study, Tsinghua University, Beijing 100084, China
wuyb14@mails.tsinghua.edu.cn
[2] Institute for Network Sciences and Cyberspace, BNRist, Tsinghua University, Beijing 100084, China
ktjia@mail.tsinghua.edu.cn

Abstract. In this paper, we propose a smart contract based searchable symmetric encryption scheme. The existing searchable symmetric encryption protocol can resist malicious servers when using the MAC algorithm; however, it is more effective only under the assumption that the server is running. If the server receives a user's money but does not provide a service to the user (or if the server shuts down after receiving the user's money), the user cannot withdraw the money paid. In addition, if the server wants to reduce computing costs, bandwidth, etc., then it may reduce the number of documents to be searched or omit part of the search results. As a result, there is no guarantee that all files have been searched. We use the Merkle tree to construct search integrity verification. Implementing search integrity verification ensures that it is nearly impossible for searchers to provide integrity verification without searching all documents. Smart contracts use computing resources effectively and help us better search the blockchain. All information is recorded on the blockchain and will not be tampered with. In addition, integrity verification and smart contracts slightly reduce the efficiency but are feasible in practice. Finally, we have theoretically and experimentally verified the safety and feasibility of the proposed scheme.

Keywords: Smart contract · Blockchain · Searchable symmetric encryption · Result integrity · Verifiable · Bloom filter

1 Introduction

A keyword index helps us search for documents containing specified keywords in a certain period of time. Security indexes are extensions of building data structures, e.g., indexes provided by unrelated data structures [13] and historically independent [2,12] data structures. Unfortunately, the standard index structure using hash tables is not suitable for indexing encrypted documents because they leak information about the contents of the document. However, data structures with privacy protection can be used to build secure indexes.

B. Chen and X. Huang (Eds.): AC3 2021, LNICST 386, pp. 181–202, 2021.
https://doi.org/10.1007/978-3-030-80851-8_14

To improve the efficiency of searchable symmetric encryption (SSE), Goh et al. proposed using the Bloom filter method to search symmetric encryption [4]. The Bloom filter proposed by Bloom is a space-saving random data structure that uses a bit array to represent a collection very concisely. It can determine whether an element belongs to this collection. The Bloom filter was used in the early UNIX spell checker [9,11]. The Bloom filter is also used as an index for each document to search the keywords [4] in each document. In the index, a keyword is represented by a codeword derived by applying a pseudorandom function twice, i.e., once using the keyword as input and once using a unique document identifier as input.

In 2017, Li et al. [8] combined blockchain technology with SSE, where a blockchain is used as a peer-to-peer network to store user data. This scheme adds data as blocks to the blockchain, thereby storing the data in the common chain. However, this scheme only supports single keyword search. Tang et al. [1] proposed two frameworks that support blockchain technology, and these frameworks can be applied to most existing SSE schemes to achieve fairness while maintaining the original privacy protection. However, these frameworks are inefficient. Zhang et al. [17] proposed a blockchain based searchable encryption scheme in multicloud environment. Here, multiple cloud service providers are combined to share data through an alliance chain. Then, the encrypted document and document index are stored in IPFs, and the hash value of the document is stored in the blockchain. This scheme can provide retrieval of outsourced encrypted data based on multiple keywords; however, it is based on trusted cloud service providers, which cannot resist malicious cloud servers.

We employ the Bloom filter to construct a searchable symmetric encryption scheme and use smart contracts to maintain the fairness of the searchable symmetric encryption scheme.

The smart contract was originally proposed by Nick Szabo in 1996 [14]. A smart contract is a computer protocol designed to facilitate, verify, or execute a contract. Note that a smart contract in the blockchain is traceable and cannot be tampered with [10]. Many contract clauses can be executed partially or completely in an independent manner. The purpose of smart contracts is to provide better security than traditional contracts and reduce other transaction costs associated with the contract. Various cryptocurrencies are implemented as types of smart contracts. A smart contract is a set of commitments specified in digital form, including an agreement for parties to fulfill these commitments [14].

Our Contribution: We exploit computing resources in the blockchain to propose a searchable symmetric encryption scheme based on smart contracts. To address incomplete retrieval of all files on the server, we first propose a non-interactive integrity verification method for search results. The proposed method uses Merkle trees to construct search integrity verification, which can ensure that it is nearly impossible for searchers to provide integrity verification without searching all documents. In addition, using smart contracts, we can use the computing resources in the blockchain effectively to perform ciphertext searches. As a result, neither the owner nor searcher can cheat. All information (including search

results, proof of integrity, and commission payment information) is recorded in the blockchain and cannot be tampered with. By implementing integrity verification and smart contracts, the proposed solution does not reduce efficiency significantly and ensures that searchers cannot obtain information from encrypted documents and search keywords. The proposed scheme guarantees the authenticity and completeness of the results, and, through the Merkle root signature, the proposed scheme ensures that searchers of search results will not be tampered with by miners after submitting the results.

The remainder of this paper is organized as follows. We define notations used in this paper in Sect. 2. In Sect. 3, we provide preliminary information about searchable symmetric encryption and smart contracts. In Sect. 4, we propose the scheme of the non-interactive integrity verification of search results for the first time. In Sect. 5, we present our scheme of smart contract based SSE and discuss the schedule in detail. In Sect. 6, we provide a security definition, security analysis, and performance analysis. Finally, the paper is concluded in Sect. 7.

2 Notations

We define the notations used in this paper in Table 1.

Table 1. Notations

Notation	Description
$\wedge$	Bitwise and
$H_1(), f()$	Pseudorandom function
n	Number of documents
m	Assume that each file has m keywords for brevity
r	Number of function $f()$ used in Bloom filter
s	Key length used in the f function
t	Key length of the encrypted file, the key length of the MAC
l	Security parameter, number of random numbers generated in the integrity verification of search results
n_{tran}	Transaction size
$H()$	Collision resistant hash function
$\|\|$	Concatenation
$bf(y)$	Obtain output of Bloom filter according to y

3 Preliminaries

We propose a searchable symmetric encryption scheme called smart contract-based SSE. The proposed scheme employs a non-interactive result integrity proof method, the Bloom filter, searchable symmetric encryption, and smart contract

technology. Here, we introduce the Bloom filter, searchable symmetric encryption and smart contract technology. Eu-jin Guo introduced a Bloom filter method to solve the problem of searchable symmetric encryption [4], which was used to solve the searchable symmetric encryption efficiency problem. We propose an improved scheme in their framework. Szabo proposed the smart contract in 1994 [15], and the proposed method employs smart contracts as a platform to increase the availability of computing resources.

3.1 Scheme of Searchable Symmetric Encryption

To improve the efficiency of searchable symmetric encryption, Eu-jin Guo introduced a Bloom filter method for searchable symmetric encryption [4] that included two participants, i.e., the file owner and the untrusted server. Here, assume the file owner has n files denoted $D_1, D_2, \ldots, D_n$. The file owner encrypts these documents into ciphertexts $C = (C_1, C_2, \ldots, C_n)$ and constructs the corresponding index set of I, and then sends C, I to the server. When the file owner wants to look up documents including keyword w, he calculates trapdoor T_w for keyword w and sends T_w to the server. The server searches for result C_i based on t_w, C and I, and then sends C_i to the file owner. Finally, the file owner decrypts C_i to obtain D_i.

Note that search symmetric encryption is considered secure if the server does not obtain any information about the plaintexts when it only knows ciphertexts or when it does not know any information about the plaintexts and keywords except the search results when it executes search algorithms.

Assume file owner U has n encryption files $C_1, C_2, \ldots, C_n$ on server $\mathcal{S}$. The SSE scheme comprises functions $(BuildIndex(\cdot), SearchIndex(\cdot))$ and three phases (**setup, search, update**). In the **setup** phase, indexes for $C_1, C_2, \ldots, C_n$ are built by the search system, the retrieval task is performed in the **search** phase, and the **update** phase updates the index when documents are changed, added, or deleted. The process of the **setup** phase is shown in Fig. 1.

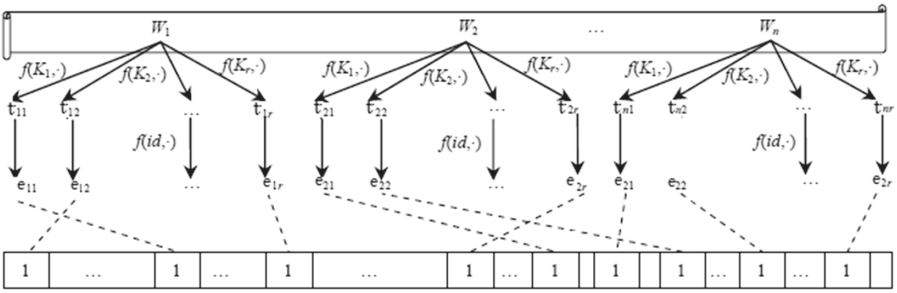

Fig. 1. The secure index setup phase

We introduce the $BuildIndex(\cdot)$ and $SearchIndex(\cdot)$ functions in the following.

- **BuildIndex($C, K_{trapdoor}$):** The function of the input is an encrypted document C with unique file ID $C_{id} \in \{0,1\}^n$ and some keywords $w = (w_0, \ldots, w_t)$, as well as the corresponding keywords trapdoor $K_{trapdoor} = (k_1, \ldots, k_r) \in \{0,1\}^{sr}$, $k_i \in \{0,1\}^s$.
  1. For each document D_i for $i \in [0, n]$, we do the following:
     (a) We compute the trapdoor of each keyword w_i, i.e. $Buildtrapdoor$ $(K_{trapdoor}, w_i) = (t_1 = f(w_i, k_1), \ldots, t_r = f(w_i, k_r)) \in \{0,1\}^{sr}$.
     (b) The corresponding ciphertext ID C_{id} information is added to get the codewords $e : e_1, \ldots, e_r$, where $(e_1 = f(C_{id}, t_1), \ldots, e_r = f(C_{id}, t_r)) \in \{0,1\}^{sr}$.
     (c) The codewords $e : e_1, \ldots, e_r$ is added to the file D_i Bloom filter BF_i.
  2. Output the index of C: $I_C = (C_{id}, BF)$.
- **SearchIndex**(T_w, I_C)**:** The function of the input is the key word trapdoor $T_w = (t_1, \ldots, t_r) \in \{0,1\}^{sr}$ corresponding to keywords W, and the index $I_C = (C_{id}, BF)$ corresponding to encryption file C.
  1. Key word w is encoded according to C_{id} to obtain e, where $e : (e_1 = f(C_{id}, t_1), \ldots, e_r = f(C_{id}, t_r)) \in \{0,1\}^{sr}$.
  2. To determine if each position in y that contains a 1 corresponds to a 1 in the Bloom filter, we must determine if $(bf(e) \wedge BF) == bf(e)$ $(e = e_1, e_2, ..., e_r)$ is correct.
  3. If the above is correct, 1 is output; otherwise, 0 is output.

The **setup**, **search**, and **update** phases are described as follows.

- **Setup phase:** The n files are uploaded to the server after the index is constructed.
  1. First, the proper Bloom filter parameters are derived for each index. In addition, we define $K_{trapdoor} = (k_1, \ldots, k_r) \in \{0,1\}^{sr}$, $k_i \in \{0,1\}^s$.
  2. Each file is assigned a unique ID, which is an integer represented by $i \in [1, n]$.
  3. Each file builds an index as $I_{C_i} \leftarrow BuildIndex(C_i, K_{trapdoor})$.
  4. After each document is compressed and encrypted using standard algorithms, the document and its index can be placed on server $\mathcal{S}$.
- **Search phase:** When file owner $\mathcal{U}$ has a lookup requirement, server $\mathcal{S}$ must be queried for the given keyword w so server $\mathcal{S}$ can return the file containing the given keyword. Then, the following is performed.
  1. File owner $\mathcal{U}$ calculates the trapdoor of w to obtain $T_w \leftarrow Buildtrapdoor(K_{trapdoor}, w)$, and then sends T_w to the server $\mathcal{S}$.
  2. For all the indexes of I_{C_i}, server $\mathcal{S}$ calls SearchIndex (T_w, I_{C_i}) to test matches. All matched files are returned to file owner $\mathcal{U}$.
- **Update phase:** There are three possible scenarios in which an update operation may be used.

1. Add file: When a file is added to the system, a unique ID is first assigned to ciphertext file C, and then index I_{C_i} is created for C.
2. Delete file: When a file is deleted from the system, we must delete both the file and its index file.
3. Changing the contents of an existing document requires assigning a new unique ID to the document and rebuilding the index by calling the $BuildIndex$ function with the new unique ID.

3.2 Smart Contract

Smart contracts are computer programs running on the blockchain. A smart contract comprises program code, stored files, and an account balance. Any user can create smart contracts by posting events to the blockchain. When creating a smart contract, the contract's program code is fixed and cannot be changed. As shown in Fig. 2, the contract storage file is stored in the blockchain. The contract's program logic is executed by miners who reach consensus on the execution results and update the blockchain accordingly. The contract code is executed when the user or another contract receives the message. When executing its code, the contract can read or write from its storage file. A contract can have an account to accept transfers from other accounts or contracts, or it can send transactions to other accounts or contracts. Conceptually, the contract can be considered a special "trusted third party." However, the party is only trusted for correctness (not privacy). Note that the entire state of the contract is visible to all nodes. Figure 2 shows a schematic diagram of a smart contract.

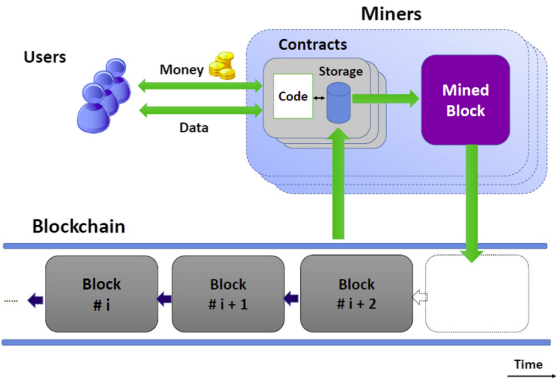

Fig. 2. Schematic of a smart contract

4 Scheme of the Integrity Verification of Search Results

Here, we introduce the non-interactive integrity verification scheme for search results. In this scheme, the Merkle tree is employed to verify the integrity of the

search results. The non-interactive integrity verification scheme can be verified on the blockchain. The searcher generates the result integrity certificate after the search is completed. The smart contract can verify whether the searcher has searched all the files according to the proof submitted by the searcher. The final submitted result is complete without any omission.

4.1 Merkle Tree

In cryptography and computer science, the Merkle tree is a tree in which each leaf node is marked with the hash value of a data block, and each non-leaf node is marked with the hash value of the label of its child nodes. Merkle trees, which are generalizations of hash tables and hash chains, allow efficient and safe verification of the contents of large data structures. To prove that a leaf node is part of a given binary hash tree, a logarithmic hash calculation of the number of leaf nodes of the tree is required. Note that this differs from hash lists because the number of hash lists is proportional to the number of leaf nodes. Figure 3 shows an example of a Merkle tree with four leaf nodes.

Merkle trees can be used to verify any type of data stored, processed, and transmitted on or between computers. They ensure that the data blocks received from other nodes in a point-to-point network are undamaged and unchanged, and can even check whether other peer nodes have tampered with the data or sent fake data. A Merkle tree is primarily used for data integrity verification based on a hash. Many systems use Merkle trees for data integrity verification, e.g., the IPFS, Btrfs, and ZFS file systems [18], as well as the Apache wave protocol [16].

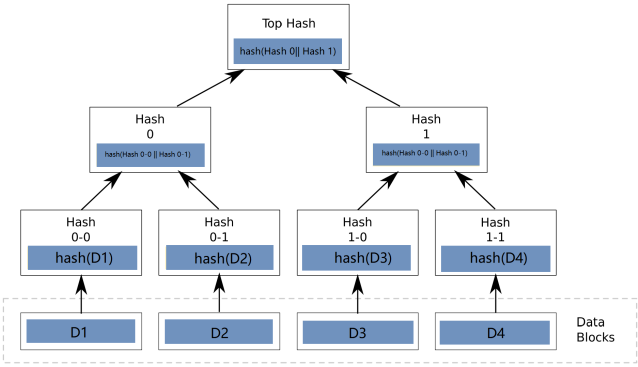

Fig. 3. Example Merkle tree with four leaf nodes

4.2 Proof of Integrity Verification of Search Results

Assuming there are n documents C, the keywords to be searched are represented by W. The file owner encrypts the keywords to trapdoor T_w and publishes

trapdoor T_w and index collections I ($I = (C, BF)$). The searcher then searches n encrypted documents according to the trapdoor T_w. The searcher calculates $bf(y)$ ($y = e_1, e_2, ...e_r$) according to T_w and document identifier C_{id}, and the search results are obtained based on $bf(y)$ and BF. To ensure the integrity of search results (that is, the searchers have searched all documents rather than only some documents), and can be use on blockchains, we propose the scheme of the non-interactive integrity verification of search results.

For the search results for the keywords in each document, we obtain the search results for each document where $search_i = (bf(y) \wedge BF_i)$ (the specific calculation process is described in Sect. 5.2.). We then use each result $search_i$ ($i = 1, 2..., n$) as a leaf node to build a Merkle tree. We obtain the corresponding root node Root when we finish constructing the Merkle tree. Here, the searcher uses their public key to sign the Merkle tree root node and obtain sign $= sign_{sk_{search}}(Root)$. Then, we use the hash function to generate a random number to construct the proof of the result's integrity. We compute $H(sign||i)$ ($i = 1, 2, ..., l$) to generate l random numbers, where l denotes the security parameter. Assuming we have n documents that need to be searched, we create an array to save a proof of the integrity of the search results, which we refer to as the array result integrity proof (RIP). We then put the $H(sign||i)$ mod n ($i = 1, 2, ..., l$) search result that is $H(sign||i)$ mod n and its path in the Merkle tree into the array RIP. We also create a Result array to save the search results. When the searcher completes the search, he outputs Result, RIP, and sign(Root). The algorithm used to search the document and construct the RIP is given in Algorithm 1.

Algorithm 1. Search document and prove integrity of search results.

Search(T_w, I)**:** When a searcher sees a demand that he wants to help search the document, then the searcher will use the trapdoor $T_w = (t_1, \ldots, t_r) \in \{0,1\}^{sr}$ for word w to test every index I_{C_i} in indexed collections I.

1: **for** every C_i in C **do**
2: The key word w is encoded according to C_{id} to get e, and e : $(e_1 = f(C_{id}, t_1), \ldots, e_r = f(C_{id}, t_r)) \in \{0,1\}^{sr}$.
3: To see if each position in y that contains a 1 corresponds to a 1 in the bloom filter, we need to detect $(bf(e) \wedge BF) == bf(e)$ $(e = e_1, e_2, ..., e_r)$ is correct, and add $(bf(y) \wedge BF_i)$ to the array search that $search_{H(sign||i) \ mod \ n}$ $(i = 1, 2, ..., l)$.
4: If so, add C_i to the array Result.
5: **end for**
6: Use array search to build a Merkle tree and get a Merkle tree root Root.
7: Searcher use his public key to sign the Merkle tree root Root and get $sign = sign_{PK_{searcher}}(Root)$
8: Compute $H(sign||i)$ $(i = 1, 2, ..., l)$, the l denote the security parameter, then add the $search_{H(sign||i) \ mod \ n}$ $(i = 1, 2, ..., l)$ and the Merkle tree path into the array result integrity proof RIP.
9: Output Result, RIP and sign(Root).

When verifying the integrity of the search results, first we verify that the signature of the Merkle tree root. We then verify that the results in the array Result are correct. Here, according to the root of the Merkle tree, we compute $H(sign||i)$ $(i = 1, 2, ..., l)$ to generate l random numbers, based on the generated random numbers and the document to compute $search_{H(sign||i) \ mod \ n}$ $(i = 1, 2, ..., l)$. Then, we verify that the path which would be provided in the array RIP of the Merkle tree of $search_{H(sign||i) \ mod \ n}$ $(i = 1, 2, ..., l)$ is correct. The algorithm used to verify results and the integrity of the results is given in Algorithm 2.

Algorithm 2. Verification of results and result integrity proof

Verify(searcher, Result, RIP, sign(Root)): This function is used to verify the result and result integrity.

1: Verify that the signature of the Merkle tree root *Root* is signed by the searcher.
2: Verify that the results in the array *Result* are correct, if correct then continues, else incorrect return error 0.
3: According to the root of the Merkle tree to compute $search_{H(sign||i) \ mod \ n}$ $(i = 1, 2, ..., l)$.
4: Verify that the path which be given in the array RIP of the Merkle tree of $search_{H(sign||i) \ mod \ n}$ $(i = 1, 2, ..., l)$ is correct, if correct then continues, else incorrect return error 0.
5: According to the compute process given in the SearchIndex function, the result of the array RIP is correct, if correct return 1, else the error returns 0.

In the following, we discuss an example (Fig. 4) to illustrate the proposed scheme. Here, assume there are four documents. Thus, when the search is complete, we obtain four search results: $search_0$, $search_1$, $search_2$, $search_3$. For each search result, $search_0 = (bf(y) \wedge BF_0)$, $search_1 = (bf(y) \wedge BF_1)$, $search_2 = (bf(y) \wedge BF_2)$, $search_3 = (bf(y) \wedge BF_3)$. By using these four search results as a leaf node to build the Merkle tree, we obtain the corresponding root node. Assume Alice is searching for the result; thus, Alice signs the root with her private key. The signature and i are joined together. We set i to 1 and 2. Thus, we obtain two values: $(sign||1)$ and $(sign||2)$. After hash and modulus, we obtain two random numbers between 0 and 3. Here, assume that $H(sign||1)$ is equal to 0, and $H(sign||2))$ is equal to 2. We save $search_0$, $search_2$ and the corresponding Merkle tree path into to the RIP array. The $search_0$, $search_2$ and the corresponding Merkle tree path are used for subsequent result integrity verification. In addition, we assume that $search_1$ is the result of the search criteria. We save the $search_1$ in the array Result. If Bob is the verifier, Bob first verifies that root is Alice's signature. Then, Bob verifies that the results in the array Result are correct; thus, Bob verifies that $search_1$ is correct. Then, Bob according to compute $H(sign||1)$ mod 4 and $H(sign||2)$ mod 4 to generate two random numbers. Here, Bob obtains the two random numbers 0 and 2. Then, Bob verifies that $search_0$, $search_2$ and the corresponding Merkle tree path are correct, which completes the validation.

Assume the search is complete with p $(0 \leq p \leq 1)$, there are n documents that need to be searched, and $l(l \leq n)$ random numbers need to be generated. If the searcher does not fully search the entire document, only $p * n$ documents are searched. When validation of the Merkle tree is set up, the probability that the searcher build a Merkle tree will be able to verify at once is p^l. The larger the l option, the lower the probability that the searcher will be able to pass validation (when the searcher does not search the entire document), and the greater the cost of not fully searching the entire document and provide correct Result, RIP, and sign(Root). Here larger l results in more complete search results.

In the following, we examine an intuitive example; we see that a searcher searches 99% of the files, l is set to 100, and the probability that the searchers are successful in building the Merkle tree is 0.366. From the above mentioned findings, it is obvious that the proposed method is extremely effective in ensuring the integrity of the search results.

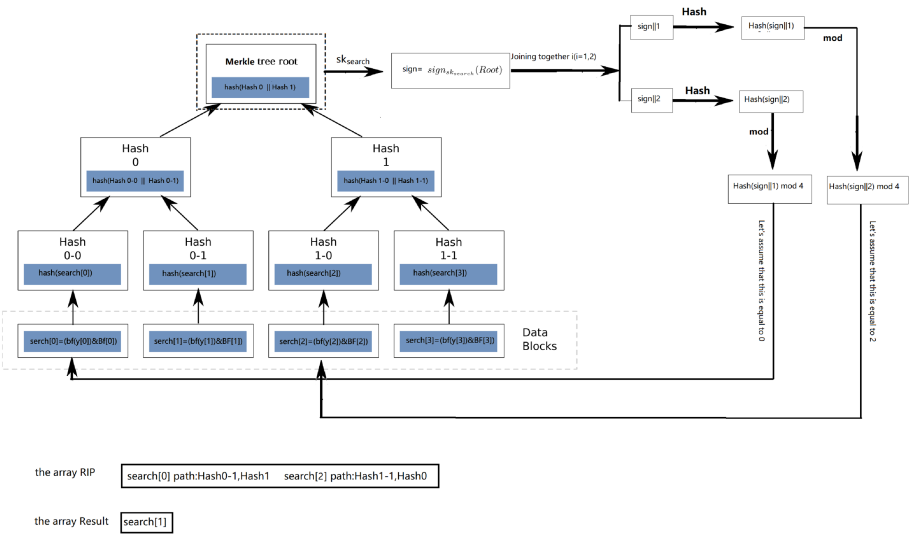

Fig. 4. Example of the proposed search result integrity verification of search results (here, assume four files to be searched and l to 2)

5 Smart Contract Based Searchable Symmetric Encryption

In this section, we propose the smart contract-based SSE scheme. The proposed scheme includes search result integrity verification algorithm, an index generation algorithms, a ciphertext search algorithm, and a demand smart contract. First, we describe the overall architecture of the proposed scheme. We then explain the proposed scheme in greater detail.

5.1 Scheme of Smart Contract Based SSE

Here, we introduce the smart contract-based SSE scheme and provide a security proof. There are three roles in the proposed scheme: file owner, searcher, and miner. The file owner has n documents denoted $D = (D_1, D_2, ..., D_n)$. The searcher's task is to search the corresponding document based on the keywords and indexes provided by the file owner. The miner's task is to verify that the search results are correct and pack the transactions into the blockchain. The owner of the file first inputs the keywords to search and provides commission d. When searcher finds searching demand then he searches the file according to the keywords and index directory, and searcher will submit search results and results integrity prove to the miners; if the verification through, miner will packaged the results and integrity proof to block chain.

Our scheme has two phases. In **series phase 1**, the file owner creates a smart contract for search demand. In **phase 2**, for a smart contract, the searcher can submit the results, and the miner can verify the results. The specific process is shown in Fig. 5, and we introduce our scheme below.

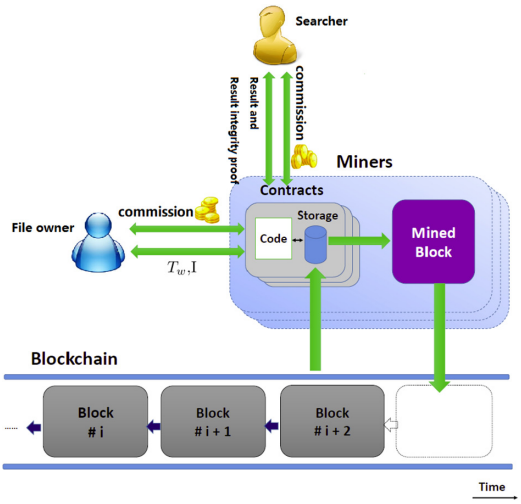

Fig. 5. Smart contract based SSE scheme

The proposed scheme involves two phases. In **phase 1**, the file owner creates a smart contract for the search demand. In **phase 2**, for a smart contract, the searcher can submit the results, and the miner can verify the results. The process is shown in Fig. 5.

In **phase 1**, the file owner wants to search keyword w, and he encrypts the keyword using the key $K_{trapdoor}$ to obtain the search trapdoor $T_w = (t_1 = f(w_i, k_1), \ldots, t_r = f(w_i, k_r))$ $(K_{trapdoor} = (k_1, k_2, ..., k_r))$, Then, the file owner

Algorithm 3. Demand smart contract

```
variable T_w, I, d, owner        \\trapdoor T_w index I = (C, BF), commissions d

function DEMAND(T_w_input, I_input, d_input)
    T_w=T_w_input
    I = I_input
    d = d_input
    owner=msg.sender
end function

function VERIFY(searcher, Result, RIP, sign(Root))
    if sign(R) is signed by searcher then
        if Result are correct then
            if the path of elements in the RIP is correct then
                count=0
                for i = 1; i <= l; i++ do
                    if search[H(sign||i)] == RIP[i] then
                        count++
                    end if
                end for
                if count==l then
                    return 1
                end if
            end if
        end if
    end if
    return
end function

function PAYCOMMISSIONS(searcher, Result, RIP, sign(Root))
    if Verify(searcher, Result, RIP, sign(R))==1 then
        Sent(owner, searcher, amount, Result)
        Selfdestruct()
    else
        return
    end if
end function

function DESTROY
    if msg.sender == owner then
        Selfdestruct()
    else
        return
    end if
end function
```

creates a smart contract that contains trapdoor T_w, index $I = (C, BF)$ and commissions d.

Algorithm 3 presents a pseudocode framework for a file owner to publish a trapdoor that needs to be looked up. Here, when creating a contract, the file owner inputs trapdoor T_w, index $I = (C, BF)$, the corresponding commission d, and the verification function.

In **phase 2**, when the searcher sees the search demand, he begins to search trapdoor T_w. When the searcher finishes searching the documents, he calls function $Verify(\cdot)$ of the demand smart contracts and uses the results, results integrity proof as parameters. If the verification is conducted through smart contract, the commissions will be sent from the file owner to the searcher, and the demand smart contract calls the function $Destroy(\cdot)$ to destroy itself. Otherwise, authentication failure is returned.

If the file owner does not want to search the document, he calls the function $Destroy(\cdot)$ to destroy the smart contract.

Search keywords can be single or multiple words. For simplicity, we only consider the single keyword case. The proposed smart contract-based SSE scheme is defined as follows:

Definition 1 *(Smart Contract based Searchable Symmetric Encryption). A smart contract based SSE is a set of six polynomial time algorithms SSE = (Gen, Enc, Build, Buildtrapdoor, Search, Demand smart contract):*

- $K \leftarrow Gen(t, s, r)$: This probabilistic algorithm takes security parameters t, s, r and outputs array key K.
- $C \leftarrow Enc(K_{file}, D)$: This algorithms takes file key K_{file}, data file collection D, and keyword collection W as input, and the file owner outputs encrypted files C.
- $I \leftarrow Build(C, W)$: This algorithms takes encrypt file C and keywords W as input, and the file owner outputs index I.
- $T_w \leftarrow Buildtrapdoor(K_{trapdoor}, w)$: This is the keyword trapdoor generation algorithm. The file owner takes search keyword w, and trapdoor key $K_{trapdoor}$ as input, and obtains trapdoor T_w as the output.
- $(result, RIP) \leftarrow Search(T_w, I)$: The searcher searches the encrypted files according to the file owner's requirements in the smart contract (keyword trapdoor T_w and index I). The search results are returned to the array Result and result integrity proof array RIP.
- Demand smart contract: This is created by the file owner and placed in the blockchain. The file owner initializes the contract and calls the constructor function Demand(·) to initialize the parameter trapdoor T_w, index $I = (C, BF)$, and commissions d parameters. The searcher calls the function PayCommissions(·) to submit the result and the RIP. The miner uses the function Verify(·) to verify the result and the RIP. The file owner calls the function Destroy(·) to destroy the smart contract.

5.2 Details of Smart Contract Based SSE Scheme

In the proposed smart contract-based SSE scheme, there are three participants, i.e., the file owner, the searcher, and the miner. The file owner has n files denoted $D = (D_1, D_2, ..., D_n)$ that need to be uploaded to the network. Prior to uploading, the file owner selects the secure function. Here, $f : \{0,1\}^* \times \{0,1\}^s \rightarrow \{0,1\}^s$ and $H_1 : \{0,1\}^* \times \{0,1\}^t \rightarrow \{0,1\}^t$ are secure pseudorandom functions. Note that $H : \{0,1\}^* \rightarrow \{0,1\}^t$ is a collision-resistant hash function, and $c = t+t+sr$ denotes the security parameter. The main steps of the algorithm are described as follows.

- **Gen(t, s, r)**. The data owner takes security parameters t, s, r as input, and it outputs secret key array $K = (K_{file}, K_{trapdoor}, K_{MAC})$, where $(K_{file}, K_{trapdoor}, K_{MAC}) \leftarrow \{0,1\}^{t+t+sr}$, $K_{file} \leftarrow \{0,1\}^t, K_{trapdoor} \leftarrow \{0,1\}^t, K_{MAC} \leftarrow \{0,1\}^{sr}$. K_{file} is used to encrypt the data documents, and $K_{trapdoor}$ is used to generate the search trapdoor T_w for the keyword w. K_{MAC} is used to generate a MAC for C_i.
- **Enc($\mathbf{K}_{file}$, D)**: There are two steps in this function. First, the data owner uses key K_{file} to encrypt the documents collection $D = (D_1, D_2, ..., D_n)$: $C_i = Enc_{K_{file}}(D_i)(1 \leq i \leq n)$, $MAC_{C_i} = H_1(K_{MAC}, C_i)$ then set $C \leftarrow ((C_1, MAC_{C_1}), (C_2, MAC_{C_2}), \ldots, (C_n, MAC_{C_n}))$. Then, the data owner generates an index to optimize the search time complexity for future searches. First, the data owner extracts keywords collection $W = (w_1, w_2, \ldots, w_r)$ from each D_i and obtains a total of nr keywords for n documents.
- **Build(C,W)**. Before the n documents are encrypted, the indexes are built as follows. Here, the input is document C comprising a unique identifier $C_{id} \in \{0,1\}^n$ (each encrypted file C_i has a unique identifier $C_{i_{id}}$), and a list of keywords $(w_1, w_2, \ldots, w_m) \in \{0,1\}^*$ (assume each file has m keywords). Here $K_{trapdoor} = (k_1, \ldots, k_r) \in \{0,1\}^{sr}$.
  I. The following is performed for each document C_i in C.
    1. For each different keyword w_j for $j \in [0, m]$, perform the following.
       (a) Compute the trapdoor of each keyword: $(t_1 = f(w_j, k_1), \ldots, t_r = f(w_j, k_r)) \in \{0,1\}^{sr}$.
       (b) The corresponding ciphertext ID C_{id} information is added to get the codeword $e : e_1, \ldots, e_r$, where $(e_1 = f(C_{id}, t_1), \ldots, e_r = f(C_{id}, t_r)) \in \{0,1\}^{sr}$.
       (c) codeword $e : e_1, \ldots, e_r$ is added to the file D_i Bloom filter BF_i.
    2. The output secure index of C is $I_C = (C_{id}, BF)$.
  II. Output $I = \{I_{C_1}, \ldots, I_{C_n}\}$
- **Buildtrapdoor($K_{trapdoor}$, w)**: Given trapdoor key $K_{trapdoor} = (k_1, ..., k_r) \in \{0,1\}^{sr}$ and keyword w, $T_w = (f(w, k_1), ..., f(w, k_r)) \in \{0,1\}^{sr}$ is output as the trapdoor.
- **Search(T_w, I)**. When a searcher observes a demand, he wants to help search the file. Then, the searcher uses trapdoor $T_w = (t_1, \ldots, t_r) \in \{0,1\}^{sr}$ for keyword w to test each index I_{C_i} in indexed collections I.

1. The following is performed for C_i in C.
   a. Keyword w is encoded according to C_{id} to obtain e, where $e : (e_1 = f(C_{id}, t_1), \ldots, e_r = f(C_{id}, t_r)) \in \{0,1\}^{sr}$.
   b. To determine if each position in y that contains a 1 corresponds to a 1 in the Bloom filter, we must determine whether $(bf(e) \wedge BF) == bf(e)$ $(e = e_1, e_2, ..., e_r)$ is correct, and then add $(bf(y) \wedge BF_i)$ to the array search that $search_{H(sign||i) \ mod \ n}$ $(i = 1, 2, ..., l)$.
   c. If this is correct, C_i is added the array Result.
2. An array search is employed to build a Merkle tree and obtain the Merkle tree root Root.
3. The searcher uses their private key to sign the Merkle tree root Root and obtains $sign = sign_{PK_{searcher}}(Root)$.
4. Compute $H(sign||i)$ $(i = 1, 2, ..., l)$ and add $search_{H(sign||i) \ mod \ n}$ $(i = 1, 2, ..., l)$ and the Merkle tree path to the array RIP, where l denotes the security parameter.
5. Output Result, RIP, and sign(Root).

– **Demand smart contract**. This contract is described in Algorithm 3. This contract contains four functions: Demand(·), Verify(·), PayCommissions(·), and Destroy(·), which are described as follows.
  – **Demand(T_w, $I = (C, BF)$, d)**. This function is used by the file owner to initialize the smart contract. The file owner initializes trapdoor T_w, index $I = (C, BF)$, commissions d and the initial contract owner.
  – **Verify(searcher, Result, RIP, sign(Root))**. This function is used to verify the result and RIP.
    1. Verify that the signature of the Merkle tree root *Root* is signed by the searcher.
    2. Check that the results in the array *Result* are correct. If the results are correct, continues; otherwise, return error 0.
    3. Compute $search_{H(sign||i) \ mod \ n}$ $(i = 1, 2, ..., l)$ according to the root of the Merkle tree.
    4. Verify that the path given in array RIP of the Merkle tree of $search_{H(sign||i) \ mod \ n}$ $(i = 1, 2, ..., l)$ is correct. If this is correct, then continues; otherwise, return error 0.
    5. According to the compute process given in the SearchIndex function, the result of the array RIP is correct. If this is correct, return 1; otherwise, return error 0.
  – **PayCommissions(searcher, Result, RIP, sign(Root))**. This function is used by the searcher to submit the lookup result and result integrity proof. If the result of the submission is verified, the smart contract transfers the commission from the file owner's account to the searcher's account. Then, the smart contract is destroyed. Here, if validation fails, a validation error is returned.
  – **Destroy()**. This function is used by the file owner to cancel the demand and destroy the smart contract. The function first determines whether the calling function is from the file owner. If so, the destroy operation is performed; otherwise, a call error is returned.

6 Security and Performance Analyses

In this section, we apply a simulation paradigm [4] to define the security of the proposed scheme. We also present a security analysis and security proof. Finally, we analyze the performance of the proposed scheme.

6.1 Security Analysis

The proposed scheme can guarantee the correctness of search data and the integrity of search results. We apply SSE based on the smart contract, and file owners can publish demands through the smart contracts. When a searcher observes the file owner's the file search demand through smart contracts, he starts to search. When the searcher finishes searching, he sends the results to the smart contracts for accuracy and integrity verification. If verification is successful, the searcher obtains the commissions. When the searcher completes the search, a Merkle tree is constructed for each file's search results, and the root of the Merkle tree is signed with the searcher's private key. Then, according to the root of the signature and i, to compute a hash, modulo the number of the files n, $H(sign||i)\ mod\ n$, and then give the search result of the corresponding $search_{H(sign||i)\ mod\ n}$ $(i = 1, 2, ..., l)$ and the path of the result in the Merkle tree. The purpose of the root signature is to ensure that when the result is submitted, the miner will not tamper with the searcher while verifying the result, and that the commission will be sent to the searcher. When the results are submitted, the searcher needs to give the results integrity proof, that are a partial search result random given by hash value and the corresponding Merkle tree path, so our scheme can avoid searcher deliberately omitting the search results to save computational cost or other reasons. The fairness of the proposed scheme is based on the fact that the blockchain cannot be tampered with. Search files on smart contracts are encrypted; thus, when only a ciphertext is retrieved, the searcher cannot learn anything about the plaintext. The search keywords in the smart contract are also encrypted; thus, the server cannot view any information about the plaintext and keywords other than the search results when executing the search algorithm. The information uploaded to the blockchain and smart contract only includes the encrypted keywords (trapdoor T_w), encrypted ciphertext, and Bloom filter; thus, no one (searchers, miners, etc.) can obtain any information related to the keywords and search plaintext through block data and the smart contract. The smart contract-based SSE follows the above two properties; thus, it provides secure SSE.

We present a theorem to prove that the proposed scheme is secure. Here, we used a real/ideal simulation model to prove security, which is fully described in the paper [4]. A number of SSE-related papers [5–7] have used this security model to prove security.

Definition 2 *(IND-CKA2 Security). If a smart contract based SSE protocol is secure, the adversary should not distinguish the real game* $Real^{\Pi}_{\mathcal{A}}(c)$ *and the simulation game* $Ideal^{\Pi}_{\mathcal{A},\mathcal{B},\mathcal{S}}(c)$.

Here, let Π be a smart contract based SSE scheme, where $\Pi = (Gen, Enc, Build, Buildtrapdoor, Search, Demand\ smart\ contract)$, and $c = t + t + sr$ is the security parameter. $\mathcal{A}$ represents the attacker, $\mathcal{B}$ represents the environment that can simulate the Ethereum system, $\mathcal{C}$ represents the challenger, and $\mathcal{S}$ represents the simulator.

$Real_A^{\Pi}(c)$: Challenger $\mathcal{S}$ can obtain key array K by performing operation $K \leftarrow Gen(j, s, r)$. After $\mathcal{S}$ obtains K, he will give the signature to be verified by the public key to Adversary $\mathcal{A}$. $\mathcal{A}$ gives challenger $\mathcal{C}$ a set of randomly selected files $D = (D_1, D_2, ..., D_n)$. Then, challenger $\mathcal{C}$ can obtain (I, C) by performing operation $Build(Enc(K, D), W)$. After challenger $\mathcal{C}$ obtains (I, C), they send (I, C) to $\mathcal{A}$. Challenger $\mathcal{C}$ returns (commissions, T_w) to attacker $\mathcal{A}$ to perform a polynomial query based on the different keyword w_i selected by the attacker $\mathcal{A}$. Then, attacker $\mathcal{A}$ queries Result and RIP by asking the $\mathcal{C}$. Finally, $\mathcal{A}$ produces a bit b that is returned by the experiment.

$Ideal_A^{\Pi}(c)$: Attacker $\mathcal{A}$ randomly selects D to satisfy condition $|D_{Ideal}| = |D_{Real}|$, where $D \leftarrow \{0,1\}^*$. Here, simulator $\mathcal{S}$ obtains (I, C) by performing operation $\leftarrow S(L(D))$ ($L(D)$ is defined in [3]). When the simulator calculates (I, C), it sends (I, C) to attacker $\mathcal{A}$. The challenger $\mathcal{C}$ returns Result and RIP to attacker $\mathcal{A}$ to perform a polynomial query in the $\mathcal{B}$ environment based on the different keyword w_i selected by attacker $\mathcal{A}$. Finally, $\mathcal{A}$ produces a bit b that is returned by the experiment.

If we can find a PPT $\mathcal{S}$ for all PPT $\mathcal{A}$, that makes $\mathcal{D}$, $|Pr[\mathcal{D}(view_{real}) = 1] - Pr[\mathcal{D}(view_{ideal}) = 1]| \leq negl(c)$ distinguishable for all polynomial sizes. Then, we can prove that Π is IND-CKA2 secure.

Theorem 1. *If the proposed SSE based smart contract scheme is an adaptive IND-CKA2 scheme, then the conditions to be met are H_1, f should be a pseudorandom function, H should be an anti-collision hash function, and $\varepsilon = (Enc, Dec)$ should be an IND-CPA [4] symmetric encryption scheme.*

Proof. We can build a PPT simulator $\mathcal{S} = \mathcal{S}_0, \mathcal{S}_1, \ldots, \mathcal{S}_q$ for attackers, which allows $\mathcal{A} = \mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_q$ to output two results $View_{real}$ and $View_{ideal}$. However these two results are indistinguishable during computation. Here, through the trace of a given history L, simulator $\mathcal{S}$ can be generated (I^*, C^*, t_w, Result, RIP) using the following methods.

- First, we discuss simulation I^*. Here, if $q = 0$, all files sizes are available to the simulator from L. $\mathcal{S}$ could set I^* to a random string of size $|I|$ by taking advantage of that information. $\mathcal{S}$ sets $C_i^* \leftarrow \{0,1\}^{|D_i|}$ first, and then sets the state of I^* to $st_{\mathcal{S}}$. Note that state $st_{\mathcal{A}_0}$ has no key value for $K_{trapdoor}$; thus, $\mathcal{S}$ will do a uniform random selection of w for each keyword.
  If $q \geq 1$, attacker $st_{\mathcal{A}}$ first select $(st_{\mathcal{A}_0}, t_{w_0^*}^*) \leftarrow \{0,1\}^*$ and ${Mac_{w_0^*}}^* \leftarrow \{0,1\}^*$ randomly and evenly for each keyword w_0^*. Then, $\mathcal{S}$ select $w_i^* (q \geq i \geq 1)$ based on $(w_{i-1}^*, st_{\mathcal{A}_{i-1}})$. $\mathcal{S}$ then randomly selects $(st_{\mathcal{A}_i}, t_{w_i^*}^*) \leftarrow \{0,1\}^*$ and ${Mac_{w_i^*}}^*$ for each keyword w_i^*.
  We know that I^* and I are indistinguishable because H_1 and $\{0,1\}^* \times \{0,1\}^t$ are indistinguishable. In the same manner, $\varepsilon = (Enc, Dec)$ is IND-CPA; thus,

we can also know that C^* is indistinguishable from the real ciphertext c. In the absence of key K_{MAC} in state st_{A_0}, there is no way for an attacker to distinguish between the ${MAC_{C_i}}^*$ and MAC_{C_i} generated in the $Enc(\cdot)$ step. This ${MAC_{C_i}}^*$ is selected randomly and uniformly from $\{0,1\}^t$.

- Now, we simulate T_w^*. If t_w^*, K_{tonke}^* and $t_w, K_{trapdoor}$ (generated by function $Build(\cdot)$) are indistinguishable, the conclusion is good. If $q = 0$, $\mathcal{S}$ will randomly and evenly select t_w^*, K_{tonke}^* from $\{0,1\}^k$. If $q \geq 1$, calculate $(t_{w_i^*}^*, K_{trapdoor}^*) \leftarrow \{0,1\}^*$ and select $w_i^*, q \geq i \geq 1$ based on $st_{\mathcal{A}_{i-1}}, w_{i-1}^*, i \geq 1$. Here, the pseudorandom functions H_1 and $\{0,1\}^* \times \{0,1\}^t$ are indistinguishable; thus, we can say that T_w^* and true T_w are indistinguishable.
- Simulate RIP^*. The pseudorandom function f is indistinguishable; thus, e and e^* generated in the search step are indistinguishable. The anti-collision hash function h is indistinguishable, the resulting RIP and the simulated RIP^* are indistinguishable.

Definition 3. *If the scheme is fair to both searchers and file owners, we say that the scheme satisfies fairness.*

Theorem 2. *If the blockchain is irreversible and the smart contract runs in the blockchain, the proposed scheme will satisfy fairness.*

- When the search task is generated, the commission is stored in the smart contract account. If the searcher fails to find the correct result, they will not obtain the commission. When the searcher finds the correct result, the file owner cannot refuse to pay the commission, and the smart contract will automatically transfer the commission to the searcher.
- The smart contract runs in the blockchain; thus, the file owner cannot change the task after publishing the search task, and the searcher cannot change the correct search result after receiving the commission.
- If both parties execute the agreement honestly, the user can obtain the correct results from the smart contract, and the searcher can obtain a commission.

Definition 4. *If we can verify whether the search results are complete and without omission, we say that the SSE scheme satisfies result integrity verification.*

Theorem 3. *If the smart contract can correctly execute our result integrity proof, then the scheme satisfies result integrity verifiable.*

Proof. After the searcher submits the results, the smart contract will implement the result integrity verification algorithm (Sect. 4.2). If the result is correct and there is no omission, the smart contract will pass verification. If the submitted result is correct but missing, it cannot pass the result integrity verification of the smart contract. □

6.2 Performance Analysis

Here, we analyze the efficiency and security of the proposed scheme and compare it to related schemes.

For users, search time determines the practicability and feasibility of the SSE scheme; thus, we primarily consider calculation and communication costs during the search stage. Table 2 shows how the proposed scheme performs in the search phase compared to related schemes. As shown in Table 2, the search time of the scheme proposed Kamara et al. [5] is the fastest because this scheme is designed to run parallel. However, this scheme does not satisfy fairness does not provide an integrity proof of the result. Thus, if the searcher returns incorrect results or if the searcher disappears after receiving a commission from the user, the user will lose both money and the search results. Therein lies the unfairness. To achieve fairness, another scheme [7] requires at least six transactions and three communications, which may prolong the time required for users to obtain results. The proposed scheme only requires two transactions and two rounds communication.

Table 2. Comparison of results of different schemes

Different scheme	Search time complexity	Communication complexity	Verifiable	Attacker	Fairness	Result integrity verifiable
Parallel [5]	O(mlog(n))	1	Yes	Server is honest-but-curious	No	No
Uc-secure [6]	O(n)	m	Yes	Server is malicious	No	No
BC [7]	O(n)	6	Yes	Server is malicious and user is malicious	Yes	No
Our scheme	O(n)	2	Yes	Server is malicious and user is malicious	Yes	Yes

6.3 Experimental Analysis

We develop an encrypted file search system. Here, we conduct relevant experiments on the number of different files and number of keywords, and we compare the time of completeness of the generated results and search time. The experimental data are shown in Table 3.

Table 3. Time required to search for encrypted files and obtain the result integrity verification

Number of documents	Number of keywords	Search time(s)	Result integrity verification(s)
2000	4	3.745	3.747
4000	4	3.912	3.902
6000	4	4.049	4.053
8000	4	4.282	4.286
10000	4	4.459	4.467
12000	4	4.576	4.571
2000	6	3.754	3.736
4000	6	3.909	3.913
6000	6	4.045	4.049
8000	6	4.283	4.287
10000	6	4.461	4.468
12000	6	4.567	4.573
2000	8	3.761	3.765
4000	8	3.908	3.917
6000	8	4.054	4.053
8000	8	4.291	4.292
10000	8	4.473	4.474
12000	8	4.577	4.581
2000	10	3.777	3.781
4000	10	3.931	3.928
6000	10	4.074	4.069
8000	10	4.296	4.293
10000	10	4.485	4.481
12000	10	4.596	4.594

From the experimental data, we conclude that, after adding the result integrity verification, the overall time is doubled, which is still within the acceptable range; thus, we consider that the proposed system provides sufficient availability.

We run our smart contract on the test network of Ethereum. Running our smart contract will burn 72000 gas per time on average, while the normal transfer on Ethereum test network will burn 20000 gas; therefore, our scheme is practical.

7 Conclusion

In this paper, we have proposed smart contract based SSE. Existing SSE protocols can resist malicious servers using MAC algorithms, but only if the server is actively running. If the server receives a user's money but does not provide service to the user or the server is closed after receiving the user's money, the user cannot withdraw the money. In addition, if the server wants to reduce computational costs and bandwidth, it will reduce the number of documents to be searched and omit a part of the search results; thus, there is no guarantee that all files will be searched. The proposed scheme solves this problem effectively using an integrity proof of search results and a smart contract. The proposed scheme guarantees the authenticity and integrity of the search results, and through the signature of the Merkle tree root that the searcher who searches the results will not be tampering by the miners after the submission of the results.

Acknowledgments. We would like to thank the anonymous reviewers. This work is supported by The National Key Research and Development Program of China (Grant No. 2018YFA0704701), National Natural Science Foundation of China (Grant No. 62072270), National Cryptography Development Fund (Grant No. MMJJ20170121), and Shandong Province Key Research and Development Project (Grant Nos. 2020ZLYS09 and 2019JZZY010133).

References

1. Androulaki, E., Karame, G., Roeschlin, M., Scherer, T., Capkun, S.: Evaluating User Privacy in Bitcoin. IACR Cryptology ePrint Archive 2012:596 (2012)
2. Buchbinder, N., Petrank, E.: Lower and upper bounds on obtaining history independence. Inf. Comput. **204**(2), 291–337 (2006)
3. Curtmola, R., Garay, J.A., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. J. Comput. Secur. **19**(5), 895–934 (2011)
4. Goh, E.-J.: Secure Indexes. IACR Cryptology ePrint Archive 2003:216 (2003)
5. Kamara, S., Papamanthou, C.: Parallel and dynamic searchable symmetric encryption. In: Sadeghi, A.-R. (ed.) FC 2013. LNCS, vol. 7859, pp. 258–274. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39884-1_22
6. Kurosawa, K., Ohtaki, Y.: UC-secure searchable symmetric encryption. In: Keromytis, A.D. (ed.) FC 2012. LNCS, vol. 7397, pp. 285–298. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32946-3_21
7. Li, H., Tian, H., Zhang, F., He, J.: Blockchain-based searchable symmetric encryption scheme. Comput. Electr. Eng. **73**, 32–45 (2019)
8. Li, H., Zhang, F., He, J., Tian, H.: A searchable symmetric encryption scheme using blockchain. arXiv preprint arXiv:1711.01030 (2017)
9. McIlroy, M.: Development of a spelling list. IEEE Trans. Commun. **30**(1), 91–99 (1982)
10. Meitinger, T.H.: Smart contracts. Informatik-Spektrum **40**(4), 371–375 (2017). https://doi.org/10.1007/s00287-017-1045-2
11. Mullin, J.K., Margoliash, D.J.: A tale of three spelling checkers. Softw. Pract. Exper. **20**(6), 625–630 (1990)

12. Naor, M., Teague, V.: Anti-persistence: history independent data structures. In: ACM Symposium on Theory of Computing, pp. 492–501 (2001)
13. Roy, S.S., Karmakar, A., Verbauwhede, I.: Ring-LWE: applications to cryptography and their efficient realization. In: Carlet, C., Hasan, M.A., Saraswat, V. (eds.) SPACE 2016. LNCS, vol. 10076, pp. 323–331. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-49445-6_18
14. Szabo, N.: Smart contracts: building blocks for digital markets. EXTROPY J. Transhumanist Thought **18**(16), 2 (1996)
15. Szabo, N.: Formalizing and securing relationships on public networks. First Monday **2**(9) (1997)
16. Wan, Z., Cai, M., Lin, X., Yang, J.: Blockchain federation for complex distributed applications. In: Joshi, J., Nepal, S., Zhang, Q., Zhang, L.-J. (eds.) ICBC 2019. LNCS, vol. 11521, pp. 112–125. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-23404-1_8
17. Zhang, C., Fu, S., Ao, W.: A blockchain based searchable encryption scheme for multiple cloud storage. In: Vaidya, J., Zhang, X., Li, J. (eds.) CSS 2019. LNCS, vol. 11982, pp. 585–600. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-37337-5_48
18. Zhang, Y., Rajimwale, A., Arpaci-Dusseau, A.C., Arpaci-Dusseau, R.H.: End-to-end data integrity for file systems: a ZFS case study. In: 8th USENIX Conference on File and Storage Technologies, San Jose, CA, USA, 23–26 February 2010, pp. 29–42. USENIX (2010)

Towards the Adaptability of Traffic-Based IoT Security Management Systems to the Device Behavior Evolutions

Chenxin Duan[1,2], Jia Li[3], Dongqi Han[1,2], Linna Fan[1,2], Shize Zhang[1,2], Jiahai Yang[1,2(✉)], and Zhiliang Wang[1,2]

[1] Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing, China
{dcx19,handq19,fln19,zsz16}@mails.tsinghua.edu.cn
[2] Beijing National Research Center for Information Science and Technology, Beijing, China
{yang,wzl}@cernet.edu.cn
[3] National Computer Network Emergency Response Technical Team/Coordination Center of China, Beijing, China
lijia@cert.org.cn

Abstract. Different kinds of Internet-of-Things (IoT) devices have been widely deployed in recent years, bringing great convenience as well as security threats. Given the grim situation of IoT security, various traffic-based security management systems specially designed for IoT systems have also been developed, such as device identification systems and anomaly detection systems. A lot of such systems are trained and evaluated on datasets collected only in short time periods and lack long-term evaluation. Intuitively, the communication behaviors and traffic profile of IoT devices may keep evolving due to factors like software or firmware update and the changes of user habits. It remains to be evaluated whether these IoT security management systems can adapt well to the device behavior evolutions, which matters a lot to the real-world performance. In this paper, we give a systematic discussion about the adaptability of IoT security management systems. We summarize the factors that may cause changes on the traffic profiles of IoT devices and how they can influence the long-term performance of IoT security management systems. We hope our work can serve as a base for further study on the building of adaptive systems for the security of IoT devices.

Keywords: Internet-of-Things · Security management · Adaptability · Device behavior evolutions

1 Introduction

The proliferation of different kinds of Internet-of-Things (IoT) devices has facilitated many aspects of people's daily life, such as smart home, smart city and

This work is supported by the National Key Research and Development Program of China (No. 2017YFB0803004).

B. Chen and X. Huang (Eds.): AC3 2021, LNICST 386, pp. 203–209, 2021.
https://doi.org/10.1007/978-3-030-80851-8_15

industrial control. However, the deployment of these mini devices also poses new challenges to cyber security. For example, Mirai, a large-scale botnet which is mainly composed of compromised IoT devices, caused great damages by launching powerful distributed denial-of-service (DDoS) attacks [2]. It is also demonstrated that such IoT-based botnets are becoming more and more resilient and can even bring down the power grid [7,13]. Given the severe security threats brought by IoT devices, many systems to perform identification [6,9,10,12], monitor [5,8,14,16] and anomaly detection [11,15] to IoT devices by traffic modeling and analysis have been developed to help network managers ensure that the deployed IoT devices are functioning as expected. We refer to these systems as IoT security management systems. Considering IoT devices usually have only very simple functionalities and constrained communication and computing capacities, many IoT security management systems depend on the characterization of traffic generated by IoT devices and the construction of models that depict the normal traffic profiles of IoT devices. Such IoT security management systems are shown to achieve excellent performance and can realize the expected security management purposes.

However, for the evaluations of these IoT security management systems, both of the training and test processes are usually based on the traffic datasets collected in short time periods. Because it is impractical to collect traffic traces spanning long time periods for the training and test of these IoT security management systems in both laboratory and real-world environments. Nevertheless, many IoT devices interact with changing environments. The changes of weather or seasons will also influence the user activities and then change the way they use IoT devices. Moreover, technical issues like software or firmware updates and configurable properties may also make the traffic profiles of IoT devices change dramaticly. Thus, a problem of adaptability, whether existing IoT security management systems can keep their high performance in the long-term running, arises because the characteristics of traffic generated by the same IoT devices may keep evolving in the process of uses.

Adaptability is of great importance for practical IoT security management systems to be deployed in real-world scenarios, without which, the systems will be very fragile and the behavior evolutions of IoT devices can easily jeopardize their performance and then compromise the network management. However, it seems that previous works on IoT security have ignored the evaluation of the system adaptability to the device behavior evolutions and this property needs more attention in the future development of IoT security management systems. In this paper, we try to give a prelimilary but systematic investigation about the problem of adaptability by discussion on the following research questions (RQ):

- RQ1: How device behavior evolutions caused by different factors will affect the traffic profiles of IoT devices?
- RQ2: Can current IoT security management systems based on the traffic profiles adapt well to the device behavior evolutions?
- RQ3: What are the possible solutions to improve the adaptability of IoT security management systems or build self-adaptive systems?

We hope our work can serve as a base for further studies on the building of adaptive systems for the security of IoT devices. The remainder of this paper is organized as follows: Sect. 2 gives a review about the existing IoT security management systems based on the traffic features or profiles; Sect. 3 summarizes and categorizes different factors that may cause evolutions on the generated traffic of IoT devices; Sect. 4 discusses the possible solutions to build adaptive security management systems for IoT devices; Sect. 5 concludes the work.

2 IoT Security Management Systems

Existing IoT security management systems based on the traffic characteristics can be divided into 3 classes: device identification, event fingerprinting and intrusion detection. We will review these 3 classes of works and the traffic features they tend to use respectively.

Device Identification: A single IoT device has only very simple functionality and there are usually various different types of IoT devices in the environments equipped with IoT systems. Thus, knowing what IoT devices are connecting to the network is the first step to enable further management policies towards the IoT devices. Besides, device identification systems can also help with the discovery of unauthorized or vulnerable devices. Some works utilize identifiable fields in the traffic to recognize IoT devices, such as destination IP addresses, domain name queries and certificates [6]. Others leverage features in different resolutions, including packet level, flow level and session level, with machine learning models to classify traffic generated by different IoT devices [9,10,12]. Many device identification methods extract feature vectors in terms of traffic traces collected in short durations, ranging from minutes to hours. Both of the selected features and the durations of instances to be classified may influence the adaptability of the device identification methods. What's more, in the evaluations of these works, the training and test datasets are usually derived from traffic generated by the same devices in different periods, completely ignoring the impact of the deployment environment and user habits, which may matter much to the real-world scenarios, especially in long terms.

Event Fingerprinting: It is presented that the events happening on IoT devices that trigger the changes of their working status (trigger events) often correspond to some fixed traffic patterns, based on which, event fingerprinting systems can be developed to help network managers monitor the working status of deployed IoT devices [8,14]. With the persistent awareness of the working status of all the IoT devices, network managers can even monitor the semantic contexts of the IoT devices and detect context-relevant anomalies like event spoofing and device failure [5,16]. Unlike device identification, event fingerprinting systems tend to use very simple features, short sequences of packet lengths and directions, to detect the events happening on IoT devices by pattern matching [5,8,14,16]. Intuitively, although such simple signatures can be very precise, they are also very fragile at the same time, because the most minor update of

the firmware or changes on distinct device configuration parameters (*e.g.*, credentials and device IDs) could destroy the signatures and it will take much effort to maintain the signatures if the changes are frequent.

Intrusion Detection: Intrusion detection systems (IDS) aim to detect all kinds of attack traffic sent by malicious attackers to compromise IoT devices. Considering the simple functionalities of IoT devices, anomaly-based IDSes, which build the profile of the normal traffic generated by IoT devices and regard any traffic deviated from the profile as intrusions, become popular in the networks serving IoT devices [11,15]. However, the traffic generated by some IoT devices may change dramaticly according to the user activities. For example, surveillance cameras are silent when users are at home and generate much large-volume traffic when the users leave their houses; the using frequency of air-conditioners can vary a lot in different seasons, which means that the communication behaviors of IoT devices can drift to deviate from the known normal profiles by themselves, not just intrusions. Without taking these factors into consideration, the long-term performance of IDSes for IoT devices may be quite questionable.

3 IoT Device Behavior Evolutions

In this section, we summarize the factors that may incur evolutions in the traffic generated by IoT devices and their possible implications on the IoT security management systems. Technical issues are the most common reasons that cause IoT devices to behave differently. Regular software or firmware updates are the most common issues that change the communication patterns of IoT devices. The IP addresses of cloud servers and domain names queried by the devices, coming from content delivery networks (CDN), can change frequently with the updates and then compromise device identification systems based on these fields. In addition, many IoT devices support multiple users, configurable credentials or parameters and user-defined trigger-action rules, which can all be reflected in the traffic characteristics. And the impact of these changes on systems based on only simple traffic features, like exact packet lengths, will be catastrophic.

Besides technical issues, the ways in which users interact with the IoT devices will also often bring evolutions to device behaviors. The functionalities of most IoT devices are closely tied with people's living and producing activities, which naturally change with the seasons. The evolutions caused by human activities, such as daily routines, lifestyles and producing plans, are mainly reflected in the using frequencies and durations of different IoT devices, which can reshape many spatial-temporal characteristics of traffic generated by the devices. For intrusion or anomaly detection systems, the changes of user habits must be taken into consideration, otherwise, by regarding traffic traces collected only in some periods as the whole normal profiles, a storm of false alerts will be generated once immense changes of user habits take place.

Having said the behavior evolutions of IoT devices, there may also exist some stable traffic features, like protocols and ports used by the devices, because the hardware and core functionalities of IoT devices can hardly change after

coming into uses. However, intuitively, these relatively stable features are not distinguishable enough to achieve the goals of IoT security management. Thus, the adaptability of IoT security management systems to the device behavior evolutions should get more attention. Both of the empirical evaluations and the enhancement of the adaptability of IoT security management systems are in highly demand in the future works.

4 Possible Solutions

In this section, we briefly discuss the possible solutions to cope with the IoT device behavior evolutions. A prelimilary method is to enrich the training datasets so that the datasets can cover all the possible traffic patterns that IoT devices may exhibit as much as possible. However, on one hand, this increases the overhead to prepare the data and makes the performance of systems sensitive to the training datasets. On the other hand, this method may only apply to the evolutions caused by user activities and the changes incurred by device software or firmware updates are usually unpredictable and remain unsolved. What's more, for anomaly detection systems, current user habits are also important factors to determine whether the ongoing device behavior is abnormal. It may hinder the system performance to simply add the training data without proper contexts.

Another possible solution is to enable lifelong learning and detection for the systems, which is a hot topic in current anomaly detection community [3]. Lifelong learning means that when the system makes some mistakes, it can be trained increamentally by directly learning from the administrators' feedback to avoid making the same mistakes again. This method is a kind of remedy afterwards with lots of human intervention and cannot eliminate the performance loss caused by evolutions proactively. Additionally, it is also challenging to make the system get aware of the different contexts between the newly coming feedback and the previously learned model that should perhaps be forgotten sometimes.

Machine learning methods are widely used in existing IoT security management systems. Nevertheless, similar problems, called *concept drift*, have already been investigated in machine learning community [1,4]. Concept drift means that the statistical properties of the target variable, which the model is trying to predict, change over time in unforeseen ways. While in IoT scenarios, many security management systems are trying to fit some mapping relationships with the traffic features, which also keep changing over time due to the device behavior evolutions. Therefore, adaptive learning methods that update predictive models online during their operation to react to concept drifts, may provide inspirations to deal with the IoT device behavior evolutions. However, a lot of future work is still needed to dive deep into the regularities of IoT device behavior evolutions and combine general adaptive learning algorithms with the IoT security management objectives.

5 Conclusion

In this paper, we focus on the adaptability of IoT security management systems to device behavior evolutions. We give systematic summary and analyses on the existing IoT security management systems, different kinds of evolutions happening on IoT devices and the possible solutions to build adaptive systems for the security of IoT devices. We find that the adaptability of IoT security management systems did not get enough attention despite its great importance and we hope our work can serve as the base of further study on adaptive IoT security management systems.

References

1. CADE: Detecting and explaining concept drift samples for security applications. In: 30th USENIX Security Symposium (USENIX Security 2021). USENIX Association, Vancouver, B.C. (2021). https://www.usenix.org/conference/usenixsecurity21/presentation/yang
2. Antonakakis, M., et al.: Understanding the mirai botnet. In: 26th USENIX security symposium (USENIX Security 2017), pp. 1093–1110 (2017)
3. Du, M., Chen, Z., Liu, C., Oak, R., Song, D.: Lifelong anomaly detection through unlearning. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, pp. 1283–1297 (2019)
4. Gama, J.A., Žliobaitundefined, I., Bifet, A., Pechenizkiy, M., Bouchachia, A.: A survey on concept drift adaptation. ACM Comput. Surv. **46**(4) (2014). https://doi.org/10.1145/2523813
5. Gu, T., Fang, Z., Abhishek, A., Fu, H., Hu, P., Mohapatra, P.: IoTGaze: IoT security enforcement via wireless context analysis. In: IEEE INFOCOM 2020-IEEE Conference on Computer Communications, pp. 884–893. IEEE (2020)
6. Guo, H., Heidemann, J.: Detecting IoT devices in the internet. IEEE/ACM Trans. Netw. **28**(5), 2323–2336 (2020)
7. Herwig, S., Harvey, K., Hughey, G., Roberts, R., Levin, D.: Measurement and analysis of Hajime, a peer-to-peer IoT botnet. In: NDSS (2019)
8. Junges, P., François, J., Festor, O.: Passive inference of user actions through IoT gateway encrypted traffic analysis. In: 2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), pp. 7–12 (2019)
9. Ma, X., Qu, J., Li, J., Lui, J.C., Li, Z., Guan, X.: Pinpointing hidden IoT devices via spatial-temporal traffic fingerprinting. In: IEEE INFOCOM 2020-IEEE Conference on Computer Communications, pp. 894–903. IEEE (2020)
10. Marchal, S., Miettinen, M., Nguyen, T.D., Sadeghi, A.R., Asokan, N.: AuDI: toward autonomous IoT device-type identification using periodic communication. IEEE J. Sel. Areas Commun. **37**(6), 1402–1412 (2019)
11. Mirsky, Y., Doitshman, T., Elovici, Y., Shabtai, A.: Kitsune: an ensemble of autoencoders for online network intrusion detection. In: Network and Distributed System Security Symposium, NDSS (2018)
12. Sivanathan, A., et al.: Classifying IoT devices in smart environments using network traffic characteristics. IEEE Trans. Mob. Comput. **18**(8), 1745–1759 (2018)
13. Soltan, S., Mittal, P., Poor, H.V.: BlackIoT: IoT botnet of high wattage devices can disrupt the power grid. In: 27th USENIX Security Symposium (USENIX Security 2018), pp. 15–32 (2018)

14. Trimananda, R., Varmarken, J., Markopoulou, A., Demsky, B.: Packet-level signatures for smart home devices. In: Network and Distributed System Security Symposium, NDSS (2020)
15. Wan, Y., Xu, K., Xue, G., Wang, F.: IoTArgos: a multi-layer security monitoring system for internet-of-things in smart homes. In: IEEE INFOCOM 2020-IEEE Conference on Computer Communications, pp. 874–883. IEEE (2020)
16. Zhang, W., Meng, Y., Liu, Y., Zhang, X., Zhang, Y., Zhu, H.: HoMonit: monitoring smart home apps from encrypted traffic. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pp. 1074–1088 (2018)

Correction to: A Novel Approach for Code Smells Detection Based on Deep Learning

Tao Lin, Xue Fu, Fu Chen, and Luqun Li

Correction to:
Chapter "A Novel Approach for Code Smells Detection Based on Deep Learning" in: B. Chen and X. Huang (Eds.): *Applied Cryptography in Computer and Communications*, LNICST 386, https://doi.org/10.1007/978-3-030-80851-8_12

In the original version of this book, a chapter title appeared with a typo. This has now been corrected.

The updated version of this chapter can be found at
https://doi.org/10.1007/978-3-030-80851-8_12

B. Chen and X. Huang (Eds.): AC3 2021, LNICST 386, p. C1, 2022.
https://doi.org/10.1007/978-3-030-80851-8_16

Author Index