Privacy-Preserving Cross-Facility Early Warning for Unknown Epidemics

Shiyu Li, Yuan Zhang, Yaqing Song
University of Electronic Science and Technology of China

Fan Wu
Central South University

Feng Lyu
Central South University

Kan Yang
The University of Memphis

Qiang Tang
The University of Sydney

Abstract

Syndrome-based early epidemic warning plays a vital role in preventing and controlling unknown epidemic outbreaks. It monitors the frequency of each syndrome, issues a warning if some frequency is aberrant, identifies potential epidemic outbreaks, and alerts governments as early as possible. Existing systems adopt a cloud-assisted paradigm to achieve cross-facility statistics on the syndrome frequencies. However, in these systems, all symptom data would be directly leaked to the cloud, which causes critical security and privacy issues.

In this paper, we first analyze syndrome-based early epidemic warning systems and formalize two security notions, i.e., symptom confidentiality and frequency confidentiality, according to the inherent security requirements. We propose EpiOracle, a cross-facility early warning scheme for unknown epidemics. EpiOracle ensures that the contents and frequencies of syndromes will not be leaked to any unrelated parties; moreover, our construction uses only a symmetric-key encryption algorithm and cryptographic hash functions (e.g., [CBC]AES and SHA-3), making it highly efficient. We formally prove the security of EpiOracle in the random oracle model. We also implement an EpiOracle prototype and evaluate its performance using a set of real-world symptom lists. The evaluation results demonstrate its practical efficiency.

1 Introduction

In human history, epidemic outbreaks have occurred on more than one occasion and have caused significant damage around the globe almost every single time. In 2009, the outbreak of H1N1 caused over 20 thousand deaths [3]. Since 2020, the COVID-19 outbreak has caused over 6 million deaths and exerted a significant adverse influence on the globe [17]. This leads us to reflect on how to reduce the damage using information technology (IT) when the next outbreak occurs.

Emerging IT architectures and systems have played a vital role in fighting against epidemic outbreaks in recent years. One prominent example is *early epidemic warning*, which identifies potential outbreaks and alerts governments as early as possible. Most existing early epidemic warning systems utilize patients' diagnosis results (i.e. their diseases) as the key evidence to identify potential epidemic outbreaks. Typical examples include the National Outbreak Report System (NORS) [13] and China Infectious Diseases Automated-Alert and Response System (CIDARS) [60]. Such *disease*-based systems have accurately warned the outbreaks of well-studied diseases (e.g., monkeypox [51]). However, they are inadequate for unknown epidemics, since the diagnosis results for an epidemic in its "nascent stage" are always erroneous.

To address the above problem, syndrome-based early warning systems are developed, where the frequencies of patients' syndromes are monitored and aberrant frequencies of syndromes (rather than diseases) serve as a trigger of epidemics' warning. Typical systems include the National Syndromic Surveillance Program (NSSP) [14, 39] and ProMED-mail [15]. Existing syndrome-based systems adopt a *cloud*-assisted paradigm (e.g., BioSense in NSSP) to achieve cross-facility statistics on the syndrome frequencies for early epidemic warning. Specifically, the cloud server collects healthcare records (including a set of symptom lists) from all the participating healthcare facilities. It then groups each symptom list *and its similar ones*, such that this group of the symptom lists essentially corresponds to one disease, compiles cross-facility statistics on the frequency of the group and launches a warning if the frequency is aberrant [57]. The syndrome-based systems gain a significant advantage in early unknown epidemic warning over the disease-based ones since patients' syndromes for unknown diseases can be quickly identified.

Security issues. Despite the advantages of the aforementioned paradigm, critical security and privacy issues arise.

• *Symptom leakage*: In many existing systems (e.g., NSSP), all the symptom contents are available to the cloud server in plaintext after removing the personal identity information (PII). However, simply removing the PII is insufficient to protect the patient's privacy when the symptom contents and other auxiliary information are known. For example, in NSSP, when a symptom list is uploaded to the cloud server, some

social information about the corresponding patient (e.g., age, race, insurance group id, and discharge date, etc.) is required along with the symptom data [6]. With the social information, the adversary can link the symptom list and the patient, identify the patient's needs and target the patient with specific advertisements to gain profits.

• *Frequency leakage*: The frequencies of symptom lists are the primary metric to generate an epidemic warning. We notice that the consequence of frequency leakage went unheeded in the past. Actually, the frequencies are derived from sensitive data generated by healthcare facilities and have significant economic value, which incentivizes some enterprises to collude with the cloud server to obtain the frequencies for (illegal) profits. This not only violates the healthcare facilities' interests, but also causes panic among the public. One notable example is inciting panic buying: as the public is sensitive to the information about public health, with symptom frequencies, it is easy for enterprises to incite the public to buy their products (such as medicines and masks in COVID-19 outbreak [8, 9]), making profits from the market turmoil, e.g., preparing for the subsequent huge demand for certain products in advance and monopolizing the market. Moreover, when the frequency of a symptom list is relatively high (but not yet to the actual alert threshold), panic would occur among people once the frequency is leaked to the public. This problem could be further exacerbated by the fact that people's behavior in a panic could disturb the social order and cause negative effects on their health [5].

Despite the existence of strict policies such as HIPAA serving as reactive measures for healthcare information leakage prevention, intentional and unintentional leakages still exist [4, 11]. Thus, technical mechanisms are required to provide proactive protections for symptoms and their frequencies.

1.1 Goals and partial solutions

Based on the above discussions about existing early warning systems, we summarize the following goals that a practical and privacy-preserving early warning system for unknown epidemics should achieve.

• Fuzzy detection. In practice, patients suffering from the same epidemic tend to have similar but not exactly identical symptom lists. Consequently, early warning systems should be able to perform fuzzy detection, i.e. *detect similar symptom lists* to monitor the frequency of each list and its similar ones.

• No symptom content or frequency leakage. As discussed before, before the frequency of a symptom list exceeds a threshold, *the symptom list and its frequency should not be leaked* to the cloud (or the healthcare facilities that have not generated the list or its similar ones).

• Low latency. The early warning system serves as the first line of the detection of potential outbreaks. If a warning is triggered in the system, further analyses on the target symptom data are initiated. It is desirable to complete this detection quickly so as to take subsequent measures as early as possible. Therefore, the early warning system should be low-latency, even when handling a substantial volume of lists.

Several techniques, including homomorphic encryption, local differential privacy, private heavy hitters, and trusted hardware, can be deployed in early warning systems for privacy preservation. We briefly discuss them below and present a detailed analysis in Section 2.

Homomorphic encryption. Homomorphic encryption is a viable approach to determine the similarity between two lists generated by different facilities without symptom leakage [36, 49]. However, a critical question arises regarding the ownership of the decryption key required to obtain the result of this fuzzy detection. Additionally, a practical challenge emerges as each symptom list necessitates comparison with all others to obtain the frequency, particularly given the substantial number of symptom lists in reality.

Local differential privacy. Local differential privacy is an emerging privacy-preserving technique for data analytics [63, 66]. However, it is unsatisfactory in the context of early warning as it would lead to a non-negligible amount of symptom and frequency information leakage.

Private heavy hitters. The solutions for the private heavy-hitters problem enable the identification of popular ones in a set of private strings [21, 27]. However, they cannot be slightly tweaked to support fuzzy detection. They also fail to achieve privacy preservation with respect to frequencies.

Trusted hardware. Hardware can be utilized to reach both the functionality and privacy goals [19, 42, 56]. Whereas, these solutions make a strong assumption about the trusted hardware. Moreover, the expensive page swaps in the hardware would slowdown the entire system.

To the best of our knowledge, there is no feasible solution to achieve the aforementioned goals simultaneously, even when putting the goal of low latency aside.

1.2 Our contributions

In this work, we take a step towards privacy-preserving early warning for unknown epidemics by constructing EpiOracle to simultaneously achieve *fuzzy detection, no symptom or frequency leakage*, and *low latency*.

Fuzzy detection. In EpiOracle, similar symptom lists are detected and mapped to one same tag, and the frequency statistics are compiled by counting the number of these tags.

No symptom or frequency leakage. To prevent symptoms from leakage, EpiOracle enables the fuzzy detection on the ciphertexts of symptom lists. To prevent frequencies from leakage, EpiOracle ensures that the status of each tag's counter is kept private from the cloud and all facilities that have not generated the corresponding symptom lists.

Low latency. EpiOracle solely relies on standard symmetric-key primitives, i.e., AES and SHA-3, and thereby is highly

	Fuzzy detection without symptom leakage	No frequency leakage	Symmetric-key cryptographic primitives only
Homomorphic encryption [36, 49]	Yes	Yes	No
Local differential privacy [63, 66]	No	No	-[1]
Private heavy hitters [21, 27]	No	No	Yes
Trusted hardware [19, 42, 56]	Yes	Yes	No
EpiOracle	Yes	Yes	Yes

Table 1: Comparisons between EpiOracle and partial solutions.

efficient without requirement on the underlying hardware.

Comparisons between previous works and EpiOracle can be found in Table 1. Our contributions are summarized below.

• We investigate the security in existing syndrome-based early warning systems and point out their inherent security requirements. We then formalize two security notions, i.e., symptom confidentiality and frequency confidentiality, for these systems, which we believe might be of independent interest to other eHealth systems.

• We construct EpiOracle, a syndrome-based early warning scheme for unknown epidemics, where the cross-facility statistics on the frequencies of symptom lists are compiled using only standard symmetric-key cryptographic primitives while neither symptom content nor frequency is leaked to unrelated parties. We further formally prove the security of EpiOracle in the random oracle model.

• We also implement an EpiOracle prototype and evaluate its performance using real-world COVID-19 syndrome data[2] as well as larger synthesized samples. The evaluation results demonstrate that EpiOracle is practical and efficient. In particular, approximately 70% of real-world COVID-19 symptom lists are detected as similar in our experiments.

1.3 Limitation discussions

One security-efficiency trade-off we chose for EpiOracle is that it allows healthcare facilities to obtain the frequencies of the symptom lists that they have generated. A corrupted facility may make those information public.

This leakage is somewhat unavoidable when we insist on no interaction between facilities in EpiOracle. Addressing this limitation (reducing further the leakage) would inherently require additional interactions among facilities to achieve that taking their symptom lists as input, only a result of whether a warning should be triggered is output. This could place a considerable burden on facilities, particularly when the number of participating entities and the volume of symptom lists are large (e.g., a partial solution could be that employing the secure comparison technique to compare a private frequency with a threshold without leaking any additional information [35, 36, 49], which requires intensive interactions among facilities). Some facilities in remote regions may not have an advanced communication infrastructure.

In this work, we aim to design an early warning system that remains practical and accessible for a wide range of facilities, including those with varying levels of communication abilities. It is impractical to expect facilities with limited communication resources to engage in intensive interactions. Therefore, we choose to adopt the non-interactive paradigm while accepting a certain degree of additional information leakage as a trade-off. It would be certainly interesting to explore other trade-offs in future works.

We also stress that any early warning system encounters certain non-technical limitations. Both our scheme and practical solutions [14, 15] for epidemic warnings are probabilistic, if the warning is not triggered, no outbreak occurs; if the warning is triggered, it only indicates the potential occurrence of an epidemic outbreak with a high probability.

1.4 Roadmap

We review the related works in Section 2 and state the problem in Section 3. We present the system model and definitions of EpiOracle in Section 4. We provide the construction of EpiOracle in Section 5, discuss practical considerations in Section 6, give formal proofs of the security in Section 7, and present the implementation details and evaluation results in Section 8. Finally, we draw the conclusions and outlook the future work in Section 9.

2 Related Work

2.1 Early epidemic warning

Early epidemic warning serves as the first line of defense against outbreaks and enables governments to gain more time to prepare for fighting against epidemics. Existing warning systems can be categorized into disease-based ones and syndrome-based ones.

[1] Instead of cryptographic primitives, differential privacy is based on mathematical operations and statistical algorithms such as adding Laplace noise.

[2] We have obtained consent from the healthcare facility to use the data for research purposes. Before being shared with us, the data has been processed by the facility to preserve patients' privacy. Personal information has been thoroughly removed, and the data only contains symptoms, ensuring that no data can be linked to patients. We have also followed academic conventions regarding this issue (e.g., Ref. [61] leverages real-world data while ensuring the utmost respect for privacy and ethical standards) to carefully release experimental results, ensuring that no patient privacy would be leaked.

Typical disease-based early warning systems include NORS [13] and CIDARS [60]. In these systems, the target diseases are specified, and a warning is triggered by confirmed cases of these diseases. Specifically, some notifiable diseases are listed by the government (the CDC in NORS, and China CDC in CIDARS), once a patient is diagnosed with a notifiable disease, a doctor reports the case in the system through the corresponding facility, and the CDC judges whether there is an outbreak by analyzing the reported data.

The disease-based paradigm can accurately identify the potential outbreaks of easily diagnosable diseases with a well-established history. However, its effectiveness diminishes when the detection of an etiology requires specialized equipment. Facilities lacking such equipment or detection capabilities must transport collected specimens to more distant facilities, resulting in delays in outbreak control. This issue is exacerbated when pathogens can only be detected within a narrow time frame after illness onset [37]. Additionally, disease-based early warning systems encounter challenges in recognizing outbreaks of unknown epidemics. Newly emerging diseases may be misdiagnosed as existing ones that do not typically trigger outbreaks. Consequently, early detection of outbreaks becomes challenging or, in some cases, fails to occur altogether.

In 2003, the CDC initiated NSSP [14], a syndrome-based early warning system. NSSP employs a cloud server, BioSense [39], to aggregate symptom information from all healthcare facilities. Early warning is facilitated by monitoring the frequencies of symptom lists. The underlying principle is that abnormally high frequencies of similar symptoms may indicate an epidemic outbreak. For instance, during the initial stages of the COVID-19 outbreak in January 2020, with approximately 15,000 confirmed cases, the vast majority exhibited symptoms such as fever, dry cough, and fatigue [17]. The cloud server, through the analysis of symptom information and tracking the frequency of similar symptom lists, can effectively detect the onset of a COVID-19 outbreak.

In contrast to disease-based systems, the syndrome-based ones such as NSSP [14] and ProMED-mail [15] enjoy a significant advantage in early warning for unknown epidemics, where the syndrome information serves as the key evidence for warnings. However, such a paradigm suffers from critical security issues regarding the symptom information due to the deployment of the cloud server [46, 53]. There exists a potential risk of the cloud server attempting to access and exploit sensitive symptom contents for personal gain [31]. Furthermore, the frequencies themselves hold significant value [32], and collusion between companies and the cloud server may lead to the exploitation of frequencies for financial gain.

2.2 Statistics over private strings

The key idea behind the privacy-preserving early unknown epidemic warning is to compile statistics on the frequencies of private data. There have been several techniques that can be deployed for the frequency statistics, and we analyze them one by one below.

Homomorphic encryption. Homomorphic encryption is a powerful primitive, which enables the performance of cryptographic operations on plaintexts through computations on corresponding ciphertexts [41, 64]. It can be utilized to determine the similarity between two private strings, e.g., computing the Hamming distance between their ciphertexts using bit-wise encryptions [36, 49]. However, deploying it in early warning suffers from several issues: 1) a private key is required for decryption to obtain the comparison result, which raises an immediate problem of which entity should hold the private key. One potential solution is to share the key among relevant facilities, which becomes cumbersome as the number of facilities increases; 2) when a symptom list is generated, all other lists should be checked, making it impractical due to the substantial number of symptom lists in practice.

Local differential privacy. Previous research has explored the use of local differential privacy to analyze private data [22, 54, 55, 63, 66], which could be adopted in early epidemic warning. Specifically, each healthcare facility individually collects its own frequencies of each symptom list[3], introduces some noise to the frequencies and then uploads the perturbed data to a central aggregator for aggregation. This approach thwarts the leakage of the frequencies *owned by each facility*. However, it inadvertently discloses the "global" frequencies of all lists to the aggregator. Furthermore, it leaks a non-negligible amount of information about symptom content.

Private heavy hitters. General-purpose private heavy hitters can also be used for early epidemic warning, where each healthcare facility holds a private symptom list, and an additional entity such as a cloud acquires popular lists without learning any additional information about any facility's list [21, 27]. While this approach has the benefit of preventing the leakage of the symptom content, the resulting protocols cannot be directly adapted to support statistics on the frequencies of similar symptom lists in early epidemic warnings. Additionally, an entity (e.g., the cloud) is required to keep track of the frequency of the heavy hitters, i.e. the popular lists, which leaks the frequencies to the entity.

Hardware-based solutions. An alternative solution is to rely on trusted hardware (e.g., Intel SGX [50]) for frequencies statistics, where the frequencies of all symptom lists are computed in a secure enclave [19, 42, 56]. Such a paradigm allows for the compilation of frequency statistics on similar lists over plaintext without symptom or frequency leakage. However, it has some inherent limitations. It makes strong security assumptions on the underlying hardware, making

[3] Given the substantial size of the potential space of symptom lists, practical implementations may employ techniques such as locality-sensitive hashing to reduce the space size [66]. In this context, each facility maps similar lists to the same string, utilizing the frequencies of these strings as input for the local differential privacy mechanism.

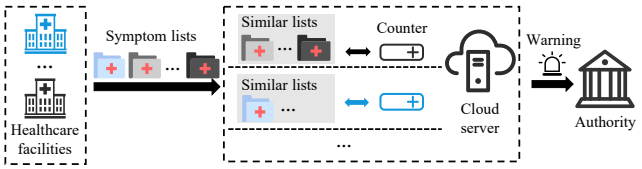


Figure 1: System model of the basic scheme.

it vulnerable to emerging attacks such as side-channel attacks [40, 47, 58, 59]. Additionally, in cases where the volume of list data is substantial, the system may experience slowdowns due to the costly page swaps resulting from limited enclave memory [34]. In contrast, EpiOracle operates without assumptions about trusted hardware, offering a distinct advantage in security considerations.

In this paper, we develop EpiOracle, a cross-facility privacy-preserving early warning system for unknown epidemics based on symmetric-key cryptographic primitives.

3 Problem Statement

3.1 Background

Notations. We denote by $s_1||s_2$ the concatenation of two strings s_1 and s_2, by $m[i]$ the i-th byte of the string m, by $\overrightarrow{v}[i]$ the i-th component of a vector $\overrightarrow{v}$, by $F(k,m)$ the evaluation of a keyed function F with the key *key* and the input m, by $r \xleftarrow{\$} S$ randomly choosing a element r from a set S, by $\{a_i\}_{i=1}^{k}$ the set $\{a_1, a_2, ..., a_k\}$, and by $\binom{n}{m}$ the number of n-combinations of an m-set, i.e., $\binom{n}{m} = m!/n!(m-n)!$.

Symptom encoding. Currently, there are some encoding standards including SNOMED CT [16], MedDRA [12], and ICD [10]. With the deployment of one standard, different descriptions for the same symptom can be mapped to the same code. For instance, dry cough and non-productive cough are encoded as 11833005 using SNOMED CT.

Same symptom and similar symptom lists. For a patient, one disease would usually cause multiple symptoms. All these symptoms form a set called a symptom list. Moreover, different patients would have different symptom lists even if they are diagnosed with the same disease. However, for one disease, the symptom lists of different patients are similar but not exactly the same. For example, the patients would suffer from symptoms including fever, cough, and so on [33]. This implies that if two different patients are diagnosed with the same disease, the Hamming distance between their (encoded) symptom lists is small.

In this paper, all symptoms are encoded using the same standard, and the similarity of two symptom lists is estimated using their Hamming distance.

3.2 Basic early epidemic warning scheme

We introduce a basic scheme to show how early warning is achieved in existing syndrome-based systems (e.g., NSSP [14] and ProMED-mail [15]). As shown in Fig. 1, all participating healthcare facilities send the encoded symptom lists of their patients to a cloud server. The cloud server groups all the symptom lists, such that the Hamming distance between any two lists in a group is smaller than a threshold, and compiles the statistics on the frequency of each disease by counting each group of the symptom lists. If some count is aberrant, the cloud server reports this aberration to an authority (e.g., CDC) to trigger a warning for a potential outbreak.

This scheme essentially utilizes a cross-facility paradigm: the cloud server collects symptom lists from multiple healthcare facilities and monitors the frequency of each list. This makes the epidemic warning more precise and timely (compared with the single-facility paradigm), since the statistics on the frequencies of lists are more general and accurate due to the basis of numerous and different-source data.

3.3 Security goals and challenges

The above basic scheme suffers from several critical security issues: it requires the cloud server to detect similar symptom lists, which implies that all symptom contents and frequencies are accessible to the cloud server. As discussed in Introduction, the symptom contents and frequencies should not be leaked to the cloud server. These concerns correspond to the following security goals that the basic scheme fails to achieve.

- Symptom confidentiality. For a symptom list, anyone who does not generate it or its similar ones, cannot obtain the content of the symptom list.
- Frequency confidentiality. Before a warning is launched by the frequency of a symptom list, anyone who does not generate the list or its similar ones, cannot obtain the frequency.

In this paper, we target to construct a cross-facility early warning scheme for unknown epidemics while achieving the above security goals, in which the following challenges exist:

- Fuzzy detection with symptom confidentiality. A natural approach to achieve symptom confidentiality is to utilize an encrypt-then-outsource paradigm: a healthcare facility encrypts a symptom list and then outsources the ciphertext to the cloud server. However, due to the inherent security of an encryption algorithm, different (even similar) lists will yield different ciphertexts. This fails to enable the cloud server to determine whether two lists are similar and thereby precludes statistics on the frequencies of the lists. Therefore, how to detect similar lists over ciphertexts is the first challenge.
- Ensuring frequency confidentiality. To achieve the cross-facility early warning, the cloud server needs to maintain a counter for each group of the symptom lists. After a list is collected from a healthcare facility, the corresponding counter is incremented. The cloud server can directly obtain the frequen-

cies of lists from the counters. This suggests an immediate solution where the counter is encrypted using homomorphic encryption, and the healthcare facility interacts with the cloud server to increase the counter, which prevents the cloud server from extracting the frequency of the symptom lists by accessing the counter. However, the access pattern of the facility's updates to counters leaks information to the cloud server by revealing the increment histogram. Therefore, how to ensure the confidentiality of frequencies is also a challenge.

3.4 Technical overview

We overview EpiOracle, focusing on the challenges addressed in this paper.

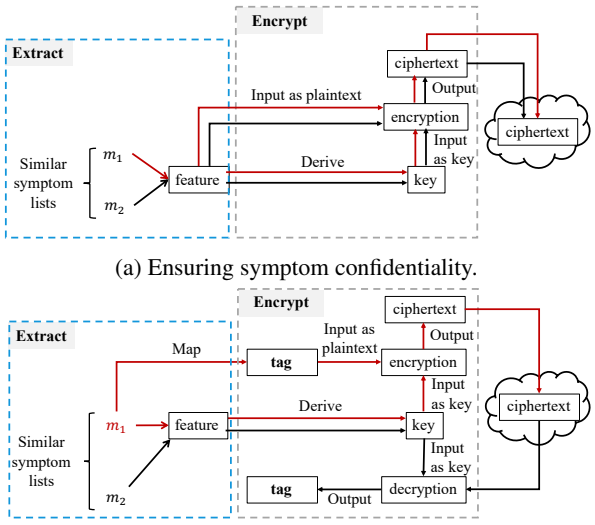
Traditional encryption ensures symptom confidentiality but precludes early warning. A natural approach to achieve confidentiality of the symptoms is to utilize an encrypt-then-outsource paradigm: after generating a symptom list, a healthcare facility encrypts the list and outsources the corresponding ciphertext to the cloud server. Directly deploying traditional encryption algorithms (e.g., [CTR]AES) in the early warning systems ensures the confidentiality of symptoms. However, due to the inherent randomness of the encryption[4], the cloud server cannot determine whether the underlying symptoms of two ciphertexts are similar. This makes the statistics on the frequencies of symptom lists impossible.

Detecting same plaintexts over encrypted symptom lists. Inspired by encrypted data deduplication, we may utilize message-locked-encryption (MLE) [24], where the encryption/decryption key is derived from the plaintext itself, to protect the symptom contents against leakage while supporting the statistics on symptom frequencies. By doing so, identical symptom lists are mapped to identical ciphertexts, which enables the cloud server to monitor the frequency of the same lists using only their ciphertexts.

Unsatisfactory for early warning. Nevertheless, straightforward utilizing MLE for the statistics on the frequencies cannot achieve the functionality of early warning in actual eHealth systems. As a reminder, in Section 3.1 we have introduced that, for the same disease, different patients would have different symptom lists. As such, the statistics on the frequency of only one symptom list will be far less than the number of patients with the corresponding disease, which cannot accurately reflect a potential outbreak.

Fuzzy detection. We address the above issue by using an extract-then-encrypt mechanism [30]. As shown in Fig. 2a, for a symptom list and its similar ones, a unique feature that can be extracted from them is encrypted under itself. By doing so, *similar* lists will generate an identical ciphertext, and the ciphertext can be outsourced to the cloud for fuzzy detection. As such, the cloud server can maintain a counter for each ciphertext to monitor the frequencies of (similar) lists.

[4]We stress that any CPA-secure encryption scheme would be randomized.



(a) Ensuring symptom confidentiality.

(b) Ensuring symptom and frequency confidentiality.

Figure 2: Extract-then-encrypt mechanism.

Ensuring frequency confidentiality. There is still a subtle security issue: the above paradigm inherently requires the cloud server to detect similar symptom lists and further count each list as well as its similar ones. This inevitably allows the cloud server to extract the frequency information. To resolve this deadlock, we utilize a facility-increased counting mechanism, where

- the fuzzy detection is migrated from the cloud server side to the facility side, and
- a facility increases the count of a symptom list with the aid of the cloud server in an oblivious way.

By oblivious, we mean that after each increment to the count of a symptom list, the cloud server cannot determine which symptom list's count is increased.

To achieve the fuzzy detection on the facility side, we improve the extract-then-encrypt mechanism introduced before. As shown in Fig. 2b, for a symptom list and its similar ones, we require the healthcare facility that generates the first one of them to map the symptom list to a tag (which is a random string), derives a key from the feature, encrypts the tag under the key, and outsources the ciphertext to the cloud server as a helper parameter for subsequent fuzzy detection. Anytime another facility generates a similar list, it first decrypts the ciphertext using a key derived from the newly-extracted feature. If and only if the feature extracted from the symptom list is the same as the one used in encryption, can the decryption succeed and the facility obtain the tag. Finally, to hide the fact that a similar list has been generated before and pass along the same tag, each subsequent facility also generates a (different) helper parameter and outsources it to the cloud server.

The oblivious increment is achieved by utilizing a *variant* of Bloom Filter [48]. Specifically, the cloud server maintains a Bloom Filter to count all symptom lists. To increase the count of a symptom list as well as its similar ones, a healthcare

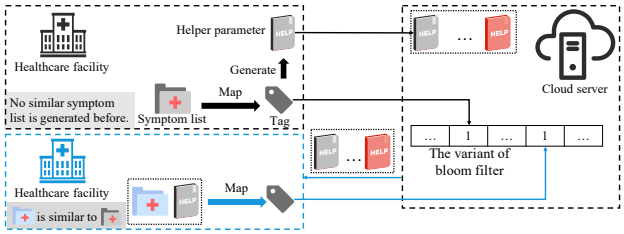


Figure 3: System model of EpiOracle.

facility randomly fills a slot of the Bloom Filter in the corresponding tag's item set[5]. The security of the variant of Bloom Filter ensures that given two increments, the cloud server cannot distinguish whether they correspond to similar symptom lists. Therefore, with this variant, in the view of the cloud server, increasing the count of a symptom list is essentially filling a random slot and nothing about the frequency can be extracted. After being increased multiple times, the count of a tag can be retrieved by computing the number of filled slots in the tag's item set. This yields EpiOracle, achieving the security goals presented in Section 3.3.

Feasibility of facility-triggered paradigm. EpiOracle adopts a facility-triggered paradigm, where the frequency of a symptom list remains private to all entities except the facilities that have generated the list or its similar ones. This raises concerns regarding the potential for facilities to conceal abnormal frequencies and maliciously trigger false warnings for arbitrary symptom lists without being detected.

However, the issues of cover-ups and false warnings can be easily addressed. Specifically, the aberrant frequency of a symptom list signifies that multiple facilities have generated the list. In such instances, even if a facility withholds the aberrant frequency, other facilities would report it. Additionally, the validity of warnings is verifiable. When a facility triggers a warning based on a frequency, it is obliged to publish the corresponding symptom list along with its tag. With the symptom list and the tag, anyone can compute the frequency of the list using the Bloom filter maintained by the cloud and further verify the validity of the warning.

4 Framework of EpiOracle

4.1 System model

As shown in Fig. 3, EpiOracle involves two entities: healthcare facilities and a cloud server. Each healthcare facility generates symptom lists, maps each list as well as its similar ones to the same tag, generates a helper parameter using the tag for subsequent fuzzy detection, outsources the helper parameter to the cloud server, and increases the count of the list by inserting the tag into a Bloom Filter. The facility can also compute the frequency of a list according to the Bloom Filter.

[5]The detailed elaboration of the variant of Bloom Filter is provided in Appendix A.

4.2 Syntax

Definition 1. *EpiOracle is a tuple of five algorithms: (**Setup**, **FuzDet**, **HelParGen**, **Increase**, **Warning**).*

$\left\langle sp, \overrightarrow{BF} \right\rangle \leftarrow$ **Setup**(1^ℓ). The **Setup** algorithm takes a security parameter ℓ as input and outputs a set of system parameters sp along with an empty Bloom Filter $\overrightarrow{BF}$. sp is implicitly input to the subsequent algorithms.

$\langle \tau \rangle \leftarrow$ **FuzDet**$(\{p_1, p_2, ...\}, m)$. After generating a symptom list m, a healthcare facility and the cloud server run the **FuzDet** algorithm to check whether some symptom lists similar to m has been generated before. It takes as input the helper parameters $\{p_1, p_2, ...\}$ maintained by the cloud server and m, and output m's tag τ. Helper parameters are generated by the following algorithm.

$\langle p = (c, z) \rangle \leftarrow$ **HelParGen**(m, τ). The **HelParGen** algorithm is run by a healthcare facility to generate a helper parameter p for subsequent fuzzy detection. It takes as input a tag τ of m, and outputs a helper parameter p consisting of τ and auxiliary information z. p is then outsourced to the cloud server.

$\left\langle \overrightarrow{BF'} \right\rangle \leftarrow$ **Increase**$(\tau, \overrightarrow{BF})$. The **Increase** algorithm is executed by a healthcare facility and the cloud server to increment a tag τ's count recorded in the Bloom Filter $\overrightarrow{BF}$. It takes as input τ and $\overrightarrow{BF}$, and outputs the updated Bloom Filter $\overrightarrow{BF'}$, in which the count of τ has been incremented.

$\langle 1/0 \rangle \leftarrow$ **Warning**$(\tau, t, \overrightarrow{BF'})$. The **Warning** algorithm is run by a healthcare facility and the cloud server to count τ and assess the frequency of the corresponding symptom list. It takes as input τ, a warning threshold t, and the Bloom Filter $\overrightarrow{BF'}$ recording all counts of the lists, outputs 1 if the count is larger than t and 0 otherwise.

4.3 Security definitions

A security definition consists of two distinct components: security goal and threat model. We have discussed the security goals (i.e., symptom and frequency confidentiality) in Section 3.2. Now we discuss the threat model and present formal security definitions of them one by one.

Symptom confidentiality. Loosely speaking, for a symptom list, ensuring its confidentiality requires that the cloud server cannot learn any additional information about its content from the information obtained by it. All information about the content of a symptom list that the cloud server can obtain is a helper parameter, which contains a tag's ciphertexts that are generated using the keys derived from the symptom list. This implies that the cloud server should not be able to distinguish the ciphertexts from random strings.

We begin with the most general chosen-plaintext-attack (CPA) model, in which the cloud server selects an arbitrary symptom list and sends it to an oracle. The oracle, in turn, selects a tag, executes **HelParGen**, and returns the resulting

$\underline{\text{IND\$-CDA}_{\mathcal{A},\mathcal{M}}(\ell)}$

$sp \leftarrow \textbf{Setup}(1^\ell)$
$m \xleftarrow{\$} \mathcal{M}$
$\tau \xleftarrow{\$} \{0,1\}^\ell$
$b \xleftarrow{\$} \{0,1\}$
$p_1 = (c_1, z) \leftarrow \textbf{HelParGen}(m, \tau)$
$c_0 \xleftarrow{\$} \{0,1\}^{|c_1|}$
$b' \leftarrow \mathcal{A}(sp, c_b, z)$
Return $(b = b')$

$\underline{\text{ROR}_{\mathcal{A}',\Pi}(\ell)}$

$m \leftarrow \mathcal{A}'^{\mathsf{E}}(1^\ell)$
$b \xleftarrow{\$} \{0,1\}$
If $b = 1$
  For $i = 1, ..., l$
    $k_i \leftarrow \mathsf{Gen}(1^\ell)$
  $c_i = \mathsf{E}(k_i, m)$
Else
  For $i = 1, ..., l$
    $c_i \xleftarrow{\$} \{0,1\}^{|\mathsf{E}(k_i,m)|}$
$b' \leftarrow \mathcal{A}'(1^\ell, \{c_i\}_{i=1}^l)$
Return $(b = b')$

Figure 4: Game defining IND\$-CDA and ROR-security.

helper parameter or a random string of the same length to the cloud server. The goal of the cloud server is to correctly guess whether the returned string is a helper parameter or a random string. In this CPA game, the cloud server can always guess correctly. Since given a symptom list and a helper parameter, the cloud server can tell whether there is a correlation between them by recovering a key from the symptom list and further decrypting the ciphertext contained in the helper parameter.

Therefore, instead of CPA, we use the chosen-distribution-attack (CDA) model defined in [24] for symptom confidentiality. As depicted in Fig. 4, the adversarial cloud server $\mathcal{A}$ obtains the distribution of the symptom lists, rather than the challenge list's content. A formal definition of the symptom confidentiality is presented below.

Definition 2. *(Symptom confidentiality.) EpiOracle is IND\$-CDA secure if, for a* β*-entropy* l*-samples source* $\mathcal{M}$*, a probabilistic polynomial-time (PPT) adversary* $\mathcal{A}$*,*

$$\text{Adv}^{\text{IND\$-CDA}}_{\mathcal{A},\mathcal{M}}(\ell) \leq 2 \cdot \text{Adv}^{\text{ROR}}_{\mathcal{A}',\Pi}(\ell) + \frac{ql}{2^\beta},$$

where $\text{Adv}^{\text{ROR}}_{\mathsf{E},\mathcal{A}'}(\ell)$ *is the advantage of a PPT adversary* $\mathcal{A}'$ *in the real-or-random game of a symmetric-key encryption* $\Pi = (\mathsf{Gen}, \mathsf{E}, \mathsf{D})$ *defined in Fig. 4,* $T_{\mathcal{A}'} \leq T_{\mathcal{A}} + con \cdot q$*,* $T_{\mathcal{A}'}$ *and* $T_{\mathcal{A}}$ *are the running time of* $\mathcal{A}'$ *and* $\mathcal{A}$*, respectively, con is a small constant, and* $q = q(\ell)$ *is a polynomial*[6].

Frequency confidentiality. Frequency confidentiality asserts the property that the count of a tag cannot be obtained by the cloud server until it reaches a predetermined warning threshold. In the threat model, we allow the cloud server to compromise some facilities to retrieve the frequency information. Our security goal is that *the adversarial cloud server cannot obtain the frequency of a symptom list that has not been generated by the compromised facilities*. The main reason why we define such a security goal is as follows.

[6]The confidentiality of symptom inherently requires that β is large enough. If β is small, the scheme is vulnerable to dictionary-guessing attacks (DGA). This vulnerability is not considered in this section. We discuss it and detail a countermeasure in Section 7.

$\underline{\text{IND-RTA}_{\mathcal{A},\mathcal{M}}(\ell)}$

$sp \leftarrow \textbf{Setup}(1^\ell)$
$m_1 \xleftarrow{\$} \mathcal{M}, \tau_1 \xleftarrow{\$} \{0,1\}^*$
$p_1 \leftarrow \textbf{HelParGen}(m_1, \tau_1)$
$b \xleftarrow{\$} \{0,1\}$
If $b = 1$
  $m_2 \xleftarrow{\$} \mathcal{M}$, s.t. $dis(m_1, m_2) \leq d, \tau_2 = \tau_1$
If $b = 0$
  $m_2 \xleftarrow{\$} \mathcal{M}$, s.t. $dis(m_1, m_2) > d, |m_2| = |m_1|$
  $\tau_2 \xleftarrow{\$} \{0,1\}^*$
$e \leftarrow \textbf{Increase}(\tau_2)$
$p_2 \leftarrow \textbf{HelParGen}(m_2, \tau_2)$
$b' \leftarrow \mathcal{A}(sp, p_1, p_2, e)$
Return $(b = b')$

Figure 5: Game defining IND-RTA.

The facility-triggered paradigm would allow a healthcare facility to obtain the frequency information of the symptom lists it generates. If an adversary compromises the facility, the latter could directly send all the information it has to the former, and thereby no security on the symptom lists from the facility is guaranteed. We therefore ask for the strongest possible security: preventing the leakage of symptom frequency against the healthcare facility that does not generate the corresponding symptom lists before. Consequently, instead of requiring all counts to be oblivious to the cloud server, we relax the definition of the frequency confidentiality to state that the cloud server cannot compute the count of a tag corresponding to a randomly chosen list. This implies that the adversary is not able to tell whether two insertions are performed on the same (unknown) tag.

We illustrate this property in the game shown in Fig. 5: if $b = 0$, two insertions are performed on one (random) tag, else, insertions are performed on two different tags. After the insertions, the corresponding positions and the generated helper parameters are given to the adversary $\mathcal{A}$. The goal of $\mathcal{A}$ is to output the correct value of b, i.e., determine whether two insertions are made on the same tag. We provide a formal definition of frequency confidentiality below.

Definition 3. *(Frequency confidentiality.) EpiOracle is IND-RTA secure if, for any PPT adversaries* $\mathcal{A}$ *and* $\mathcal{A}'$*,*

$$Adv^{\text{IND-RTA}}_{\mathcal{A},\mathcal{M}}(\ell) \leq 2 \cdot Adv^{\text{IND\$-CDA}}_{\mathcal{A}'}(\ell),$$

where $T_{\mathcal{A}'} \leq T_{\mathcal{A}} + con$*,* $T_{\mathcal{A}'}$ *and* $T_{\mathcal{A}}$ *are the running time of* $\mathcal{A}'$ *and* $\mathcal{A}$*, respectively, and con is a small constant.*

5 Concrete Construction of EpiOracle

A cloud server $\mathcal{CS}$ and a set of healthcare facilities $\{\mathcal{H}_1, \mathcal{H}_2, ..., \mathcal{H}_N\}$ are involved in EpiOracle.

Assumption. We assume that for each $i \in [1,N]$, there is a secure channel between $\mathcal{H}_i$ and $\mathcal{CS}$ for communication. The messages transmitted through these channels are authenticated and cannot be tampered with.

Setup. With the security parameter ℓ, the system parameter $sp = \{\overrightarrow{BF}, L, t, l, \alpha, P, h_1, ..., h_s, H, \Pi\}$ is determined, where $\overrightarrow{BF}$ is an L-bit vector initialized as an empty Bloom Filter, t is the warning threshold, l is the number of rounds in helper parameter generation, α is the number of positions sampled from a symptom list, $P \in \{0,1\}^\ell$ is a random value, $h_1, ..., h_{s-1} : \{0,1\}^* \to \{0,1\}^L, H : \{0,1\}^* \to \{0,1\}^\ell$ are hash functions, $\Pi = (\mathsf{Gen}, \mathsf{E}, \mathsf{D})$ is a symmetric-key encryption scheme, $\mathsf{E}(K,M)$ is encrypting M under K, and $\mathsf{D}(K,C)$ is decrypting the ciphertext C using the decryption algorithm.

FuzDet. After generating some symptom lists, $\mathcal{H}_i$ first checks whether they are similar over the plaintexts, clusters similar lists, and chooses one from each cluster to run this algorithm. For the sake of brevity, we assume $\mathcal{H}_i$ chooses m_i from a cluster contained ε lists and show how $\mathcal{H}_i$ checks whether m_i is similar to $m_{i'}$ that has been generated before[7].

- For existing help parameters p_j of m_j, $\forall j \in [1,i]$, public stored on $\mathcal{CS}$, $\mathcal{H}_i$ checks if there is a $p_{i'}$ of $m_{i'}$ ($i' \in [1,i]$) such that $m_{i'}$ is similar to m_i by invoking Algorithm 1.
- If returned $\tau_{i'} = \perp$, $\mathcal{H}_i$ randomly chooses $\tau_i \in \{0,1\}^\ell$ as the tag of m_i, otherwise, sets $\tau_i = \tau_{i'}$.

HelParGen. $\mathcal{H}_i$ generates a help parameter p_i of m_i for subsequent similar lists by invoking Algorithm 2.

Increase. In this algorithm, $\mathcal{H}_i$ increases the count of τ_i by inserting it into the Bloom Filter $\overrightarrow{BF}$ as follows.

- $\mathcal{H}_i$ computes the set of τ_i's slots as $V_i = \{h_k(\tau_i), \forall k \in [1,s]\}$, randomly chooses $e \in V_i$ such that $\overrightarrow{BF}[e]$ is empty, sends it to $\mathcal{CS}$ (If no element meets this condition, $\mathcal{H}_i$ aborts).
- After receiving e, $\mathcal{CS}$ checks whether $\overrightarrow{BF}[e]$ are empty, if it is, $\mathcal{CS}$ updates $\overrightarrow{BF}$ to $\overrightarrow{BF}'$ by setting $\overrightarrow{BF}[e]$ to 1. Otherwise, $\mathcal{CS}$ asks $\mathcal{H}_i$ to choose another element.

In the above description, we detail the process that $\mathcal{H}_i$ increments the count of τ_i by 1 by filling one slot. $\mathcal{H}_i$ can increment the count by a large number by filling multiple slots.

Warning. In this algorithm, $\mathcal{H}_i$ can launch an early warning for m_i by computing its approximate frequency.

[7] The smaller the Hamming distance between two symptom lists, the more likely they are to be detected as similar, but the Hamming distance does not appear explicitly in the scheme.

Algorithm 1: FuzDet

Input: $\{p_j\}_{j=1}^{i-1}$: all existing helper parameters
m_i: the symptom list to be checked
Output: $\tau_{i'}$: the tag of $m_{i'}$ similar to m_i

for $j = 1$ **to** $i-1$ **do**
  parse $p_j = \{p_{j,1}, p_{j,2}, ..., p_{j,l}\}$;
  for $k = 1$ **to** l **do**
    parse $p_{j,k} = \{c_{j,k}, ind_{j,k}^{(1)}, ..., ind_{j,k}^{(\alpha)}\}$;
    $seed_{j,k} = m_i[ind_{j,k}^{(1)}]||\cdots||m_i[ind_{j,k}^{(\alpha)}]$;
    $key_{j,k} = H(P||seed_{j,k})$;
    parse $c_{j,k} = \{c_{j,k}^{(1)}, c_{j,k}^{(2)}\}$;
    if $\mathsf{D}(c_{j,k}^{(2)} \oplus key_{j,k}, c_{jk}^{(1)}) \neq \perp$ **then**
      return $\tau_{i'} = \mathsf{D}(c_{j,k}^{(2)} \oplus key_{j,k}, c_{jk}^{(1)})$;
    end
  end
end
return $\tau_{i'} = \perp$;

Algorithm 2: HelParGen

Input: $m_i = m_i[1]||\cdots m_i[b]$: a symptom list
τ_i: a corresponding tag
Output: p_i: a corresponding help parameter

for $k = 1$ **to** l **do**
  randomly choose $ind_{i,k}^{(1)}, ..., ind_{i,k}^{(\alpha)} \in [1,b]$;
  $seed_{i,k} = m_i[ind_{i,k}^{(1)}]||\cdots||m_i[ind_{i,k}^{(\alpha)}]$;
  $key_{i,k} = H(P||seed_{i,k})$;
  randomly choose $r_{i,k} \in \{0,1\}^\ell$;
  $c_{i,k}^{(1)} = \mathsf{E}(r_{i,k}, \tau_i), c_{i,k}^{(2)} = r_{i,k} \oplus key_{i,k}$;
  $c_{i,k} = \{c_{i,k}^{(1)}, c_{i,k}^{(2)}\}, p_{i,k} = \{c_{i,k}, ind_{i,k}^{(1)}, ..., ind_{i,k}^{(\alpha)}\}$;
end
return $p_i = \{p_{i,1}, ..., p_{i,l}\}$;

- Compute $V_i = \{h_k(\tau_i), \forall k \in [1,s]\}$ and set $count = 0$. For each $v_i \in V$, check whether $\overrightarrow{BF}[v_i] = 1$, if it is, set $count = count + 1$.
- If $count \geq T$ (the expected number of filled elements of V_i after inserting m_i into $\overrightarrow{BF}$ t times, which is computed using t and other system parameters, the computing method is detailed in Theorem 1), there might be a potential outbreak. Then, $\mathcal{H}_i$ triggers a warning.

We also illustrate the workflow of EpiOracle and its instantiation in Figure 6.

Value of T. When computing the frequency of a tag, the number of insertions performed on the tag may be different from the number of its filled slots due to the inherent approximate nature of Bloom Filters. We present how to compute the

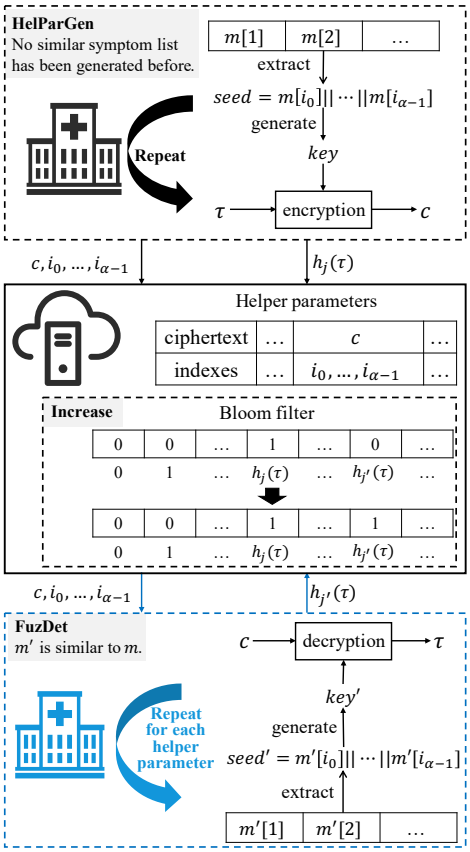


Figure 6: Workflow and an instantiation of EpiOracle.

explicit threshold T, which is the expected number of filled slots of a tag τ after t insertions performed on τ.

Theorem 1. *Given a tag τ, let ι denote the number of the insertions performed on other tags*[8]*, w denote the number of unfilled positions of $\overrightarrow{BF}$ in τ's item set, then*

$$T = s - \sum_{w=t+1}^{s} \frac{s!(L-s)!\iota!(L-\iota)!}{w!(s-w)!(L-\iota-w)!L!}(w-t).$$

Proof. Define p_w to be the probability that w slots of τ are unfilled after ι insertions performed on other tags, then

$$p_w = \frac{\binom{s}{s-w} \cdot \binom{L-s}{\iota-s+w}}{\binom{L}{\iota}}.$$

Define $R_{w,u}$ to be the number of unfilled slots of τ after u insertions performed on τ when originally w slots of τ are unfilled, then we have

$$R_{w,u} = \begin{cases} 0, & w = 0 \\ w, & u = 0 \\ w-u, & w,u \geq 1 \end{cases},$$

this can be simplified to

$$R_{w,u} = \begin{cases} 0, & w \leq u \\ w-u, & w > u \end{cases}.$$

After t insertions are performed on τ and ι insertions are performed on all tags, the expected count of τ is

$$T = s - \sum_{w=0}^{s} p_{w,t} R_{w,t}.$$

This concludes the proof. □

6 Practical Considerations

There are some practical considerations for the deployment of EpiOracle in reality. We discuss them in the following.

Use frequencies for early warning. The early warning provided by EpiOracle is compatible with the analyses provided by experts. Any early warning system is the first line of fight potential outbreaks. A triggered warning serves more as a signal that further analyses should be conducted, rather than a definitive assertion of an emerging epidemic outbreak. Moreover, EpiOracle essentially counts the symptom lists for frequency monitoring and can be trivially extended to support more complex statistics algorithms (e.g., CUSUM [20]) applied in real-world detection systems such as EARS [43].

Setting system parameters. In EpiOracle, t (the threshold) and α (the number of positions sampled from a symptom list) should be set carefully to ensure accuracy of early warning.

Value of t. Setting the warning threshold t has been well studied in the medical field [1]. It is computed according to the frequencies in the historical time period when no outbreak occurs and can be determined by infectious medical experts.

Value of α. In the generation of p_i in **HelParGen**, each seed is derived from the concatenation of random α positions of the corresponding symptom list m_i. To determine the similarity between m_i and a subsequently generated list $m_{i'}$ using p_i, these α positions of $m_{i'}$ must match those of m_i exactly. The value of α is crucial to the accuracy of the early warning and should be determined based on the length of symptom lists. For example, if α is much smaller than the length of m_i, there is a high probability of false similarity, which occurs when two lists with a large Hamming distance are incorrectly detected as similar and have the same tag.

However, the length of a patient's symptom list during an outbreak is unpredictable. This unpredictability makes it challenging to preselect an appropriate value for α. A solution to this issue is to compute a specific α for each symptom list after it is generated. Specifically, in the setup of EpiOracle, we set a deterministic ratio $SimRatio \in (0,1]$ instead of the number of chosen bytes. After m_i is generated, the number of its positions used to generate the keys is computed as $\alpha = SimRatio \cdot |m_i|$. This approach ensures that α is proportional to the length of m_i and compatible with lists of all lengths.

[8] ι can be computed by $\mathcal{CS}$ as follows: $\mathcal{CS}$ maintains a counter for the total number of insertions and subtracts t from the total number of insertions.

Different orders of symptoms. The smaller the Hamming distance between two symptom lists, the more likely they are to have the same tag. However, defining similarity based on Hamming distance may be too rigid. Consider two symptom lists: s_1 = "Dry cough, Fever, Chill" and s_2 = "Headache, Dry cough, Fever". From an early earning perspective, these two lists are similar and should have the same tag. Whereas, the Hamming distance between their codes is relatively long.

The reason for this conflict is that two symptom lists with the same symptoms but different orders have a large Hamming distance. To eliminate this conflict, we not only compare the original symptom lists but also all possible permutations of the symptoms. After re-ordering s_2 to "Dry cough, Fever, Headache", the Hamming distance between the codes of these two symptom lists is only 6. Although considering all permutations increases the computation costs, it is feasible in practice since the number of symptoms in each list is small, and the number of permutations is not too large. The improved accuracy justifies the added computation costs for facilities.

Geographic area. In reality, epidemic early warning necessarily requires the frequencies in a specific area, since people are much more likely to contract a disease if they are exposed to each other. EpiOracle can be directly extended to support specific-area frequency monitoring as follows. The cloud server maintains a Bloom Filter for each specific area, and the count of symptom lists from an area is increased using the corresponding Bloom Filter.

When EpiOracle is deployed nationwide, it seems that the computation and communication costs on the healthcare facility side would be significantly high. Since all symptom lists nationwide are uploaded to a cloud, and a facility has to detect similarities between lots of lists. However, even if EpiOracle is deployed nationwide, e.g., all healthcare facilities in the U.S. using EpiOracle, a facility only needs to perform the fuzzy detection on the lists generated in a relatively small geographic area. Furthermore, early warning is time-sensitive, only the lists generated in a short period remain in the system for tallying. This further reduces the number of symptom lists that the facility needs to process.

Verifying the validity of the recovered tag. For a symptom list with a helper parameter, a healthcare facility holding another one checks whether these two symptom lists are similar by decrypting each ciphertext of the tag in the helper parameter. To perform fuzzy detection, the facility needs to tell whether the decryption succeeds. However, in the view of the facility, a wrong decryption result may be indistinguishable from the correct tag due to the randomness of the tag. As a result, the facility may not be able to determine whether the two symptom lists are similar.

This issue can be addressed by padding the tag using a specified standard such as PKCS#5 [44]. After a tag is padded, whether the decryption of the encrypted tag is valid can be verified by checking the correctness of the padding. If the padding pattern is inconsistent with the preset one, the healthcare facility can observe that the decryption fails. Moreover, in the practical implementation, if a block cipher is used as the symmetric-key encryption algorithm to encrypt the tag, the tag is required to be padded by the algorithm. When the length of a tag to be encrypted is not a multiple of the block length of the encryption algorithm, the tag will be padded by the encryption algorithm and no additional padding is needed. When the length of the tag is a multiple of the block length, padding is still required.

Number of recovered tags. In **FuzDet**, after recovering one correct tag, a healthcare facility increases the count of only this tag and stops performing similarity checks using other helper parameters. In practice, multiple symptom lists would be similar to the one held by the facility. This one-success-then-stop paradigm is efficient in terms of the computation costs on the facility side. However, it may cause inaccuracy in the counts of other similar symptom lists.

An alternative paradigm is to require the facility to perform similarity checking using all helper parameters and to increase the counts of all recovered tags. This will cause more computation costs on the facility side, but the accuracy of fuzzy detection would be improved significantly. Actually, a threshold can be set to balance the trade-off between efficiency and accuracy: after recovering the threshold number of tags successfully, the facility stops detecting.

System refresh. In EpiOracle, the tags of symptom lists are inserted into a variant of Bloom Filter. The variant of Bloom Filter can be full after too many insertions are performed, and the counts of symptom lists could not be increased.

This will not impact the practicability of EpiOracle since counting the symptom lists generated recently is an inherent requirement of epidemic early warning systems. This implies that in the practical application of EpiOracle, only the insertions performed in a short period of time are needed (for COVID-19, 3 weeks is enough [62]). Otherwise, the count of a symptom list will increase over time, and it will always reach the warning threshold. Therefore, the Bloom Filter would be refreshed before it is full.

7 Security Analysis

Theorem 2. *If H is modeled as a random oracle HO, EpiOracle is IND$-CDA secure for symptom content.*

Proof. To prove **Theorem 2**, we first introduce 6 games presented in Fig. 7 and elaborate on them below.

Let $(seed_1, seed_2, ..., seed_l, z) \xleftarrow{\$} \mathcal{S}(1^\ell)$ denote a polynomial-time algorithm that generates l seeds using a random symptom list chosen from $\mathcal{M}$. This algorithm on input ℓ returns l random seeds and some auxiliary information z (i.e., the indexes in helper parameters), then G_1

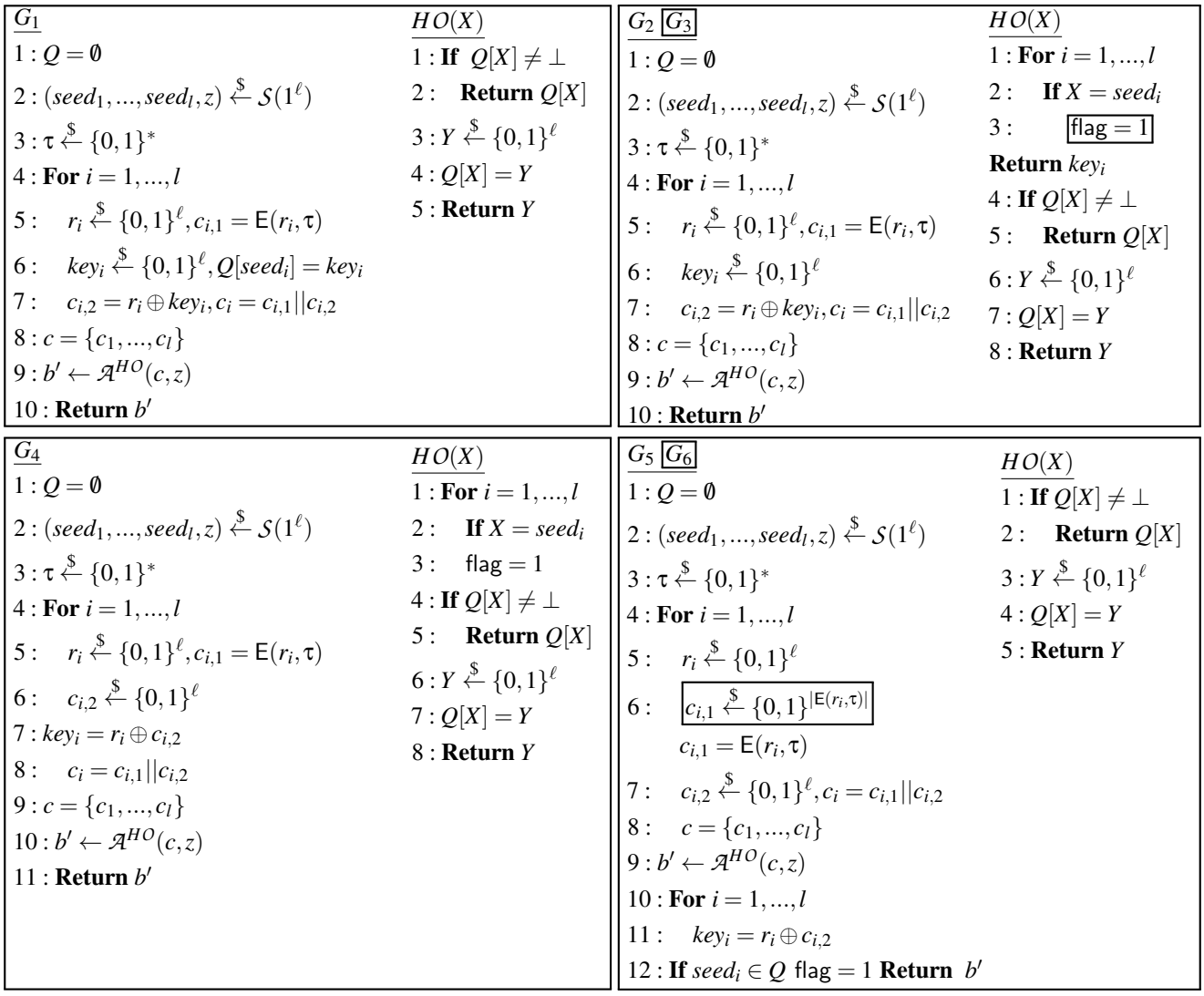


Figure 7: Game used in the proof of Theorem 2.

is identical with IND$-CDA$^{b=1}_{\mathcal{M},\mathcal{A}}$. Therefore, we have

$$\Pr[\text{IND\$-CDA}_{\mathcal{A},\mathcal{M}} \Rightarrow 1 | b = 1] = \Pr[G_1 \Rightarrow 1].$$

G_2 is identical with G_1 except that the former does not update Q with regards to $seed_i$ and key_i until $\mathcal{A}$ queries $seed_i$ to HO. Since this deferment is invisible to $\mathcal{A}$, we have

$$\Pr[G_1 \Rightarrow 1] = \Pr[G_2 \Rightarrow 1].$$

In G_3, when there is a query on $seed_i$ to HO, a flag is set to 1 and a newly chosen random string rather than key_i is returned. G_3 and G_2 are identical-until-flag. With the fundamental lemma of game-playing (Lemma 1 in [25]), we have

$$\Pr[G_2 \Rightarrow 1] \leq \Pr[G_3 \Rightarrow 1] + \Pr[\mathsf{flag} = 1 \text{ in } G_3].$$

In G_4, $c_{i,2}$ is replaced with a random string and $key_i = c_{i,2} \oplus r_i$. By doing so, c is random and thereby independent of the symmetric keys $\{r_1, ..., r_l\}$. We have

$$\Pr[G_3 \Rightarrow 1] = \Pr[G_4 \Rightarrow 1],$$
$$\Pr[\mathsf{flag} = 1 \text{ in } G_3] = \Pr[\mathsf{flag} = 1 \text{ in } G_4].$$

G_5 is identical with G_4 except that the computation of key_i and the setting of flag are deferred after $\mathcal{A}$ outputs b' in G_5. Since c is independent of $\{r_1, ..., r_l\}$, we have

$$\Pr[G_4 \Rightarrow 1] = \Pr[G_5 \Rightarrow 1],$$
$$\Pr[\mathsf{flag} = 1 \text{ in } G_4] = \Pr[\mathsf{flag} = 1 \text{ in } G_5].$$

In G_6, $c_{i,1}$ is replaced with a random string.
Now we prove that

$$\Pr[G_5 \Rightarrow 1] + \Pr[\mathsf{flag} = 1 \text{ in } G_5]$$
$$= \Pr[G_6 \Rightarrow 1] + \Pr[\mathsf{flag} = 1 \text{ in } G_6] + 2 \cdot \mathrm{Adv}^{\mathrm{ROR}}_{\mathcal{A}',\Pi}(\ell),$$

$\underline{G_7}$

$Q = \emptyset$

$m_1 \xleftarrow{\$} \mathcal{M}, \tau_1 \xleftarrow{\$} \{0,1\}^\ell$

For $x = 1,2$

  For $i = 1,...,l$

    $z_{x,i} = \{ind_{x,i}^{(j)}\}_{j\in[1,\alpha]} \xleftarrow{\$} [1,|m_1|]$

    $seed_{x,i} = m_1[ind_{x,i}^{(1)}]||\cdots||m_1[ind_{x,i}^{(\alpha)}]$

    $r_{x,i} \xleftarrow{\$} \{0,1\}^\ell, c_{x,i}^{(1)} = \mathsf{E}(r_{x,i}, \tau_1)$

    $key_{x,i} \xleftarrow{\$} \{0,1\}^\ell, Q[seed_{x,i}] = key_{x,i}$

    $c_{x,i}^{(2)} = r_{x,i} \oplus key_{x,i}, c_{x,i} = c_{x,i}^{(1)}||c_{x,i}^{(2)}$

  $c_x = \{c_{x,1},...,c_{x,l}\}, z_x = \{z_{x,1},...,z_{x,l}\}$

$b' \leftarrow \mathcal{A}^{HO}(c_1, z_1, c_2, z_2)$

$\underline{HO(X)}$

If $Q[X] \neq \perp$ **Return** $Q[X]$

$Y \xleftarrow{\$} \{0,1\}^\ell, Q[X] = Y$ **Return** Y

$\underline{G_8}$

$Q = \emptyset$

$m_1 \xleftarrow{\$} \mathcal{M}, \tau_1 \xleftarrow{\$} \{0,1\}^\ell$

For $i = 1,...,l$

  $z_{1,i} = \{ind_{1,i}^{(j)}\}_{j\in[1,\alpha]} \xleftarrow{\$} [1,|m_1|]$

  $z_{2,i} = \{ind_{2,i}^{(j)}\}_{j\in[2,\alpha]} \xleftarrow{\$} [1,|m_1|]$

  $seed_{1,i} = m_1[ind_{1,i}^{(1)}]||\cdots||m_1[ind_{1,i}^{(\alpha)}]$

  $r_{1,i} \xleftarrow{\$} \{0,1\}^\ell, c_{1,i}^{(1)} = \mathsf{E}(r_{1,i}, \tau_1)$

  $key_{1,i} \xleftarrow{\$} \{0,1\}^\ell, Q[seed_{1,i}] = key_{1,i}$

  $c_{1,i}^{(2)} = r_{1,i} \oplus key_{1,i}, c_{1,i} = c_{1,i}^{(1)}||c_{1,i}^{(2)}$

$c_1 = \{c_{1,1},...,c_{1,l}\}, c_2 \xleftarrow{\$} \{0,1\}^{|c_1|}$

$z_1 = \{z_{1,1},...,z_{1,l}\}, z_2 = \{z_{2,1},...,z_{2,l}\}$

$b' \leftarrow \mathcal{A}^{HO}(c_1, z_1, c_2, z_2)$

$\underline{HO(X)}$

If $Q[X] \neq \perp$ **Return** $Q[X]$

$Y \xleftarrow{\$} \{0,1\}^\ell, Q[X] = Y$ **Return** Y

$\underline{G_9}$

$Q = \emptyset$

$m_1, m_2 \xleftarrow{\$} \mathcal{M}, s.t.\ m_1 \neq m_2, |m_1| = |m_2|$

$\tau_1, \tau_2 \xleftarrow{\$} \{0,1\}^*$

For $x = 1,2$

  For $i = 1,...,l$

    $z_{x,i} = \{ind_{x,i}^{(j)}\}_{j\in[1,\alpha]} \xleftarrow{\$} [1,|m_1|]$

    $seed_{x,i} = m_x[ind_{x,i}^{(1)}]||\cdots||m_x[ind_{x,i}^{(\alpha)}]$

    $r_{x,i} \xleftarrow{\$} \{0,1\}^\ell, c_{x,i}^{(1)} = \mathsf{E}(r_{x,i}, \tau_1)$

    $key_{x,i} \xleftarrow{\$} \{0,1\}^\ell, Q[seed_{x,i}] = key_{x,i}$

    $c_{x,i}^{(2)} = r_{x,i} \oplus key_{x,i}, c_{x,i} = c_{x,i}^{(1)}||c_{x,i}^{(2)}$

  $c_x = \{c_{x,1},...,c_{x,l}\}$

$\underline{G_9}$

$z_x = \{z_{x,1},...,z_{x,l}\}$

$b' \leftarrow \mathcal{A}^{HO}(c_1, z_1, c_2, z_2)$

$\underline{HO(X)}$

If $Q[X] \neq \perp$ **Return** $Q[X]$

$Y \xleftarrow{\$} \{0,1\}^\ell, Q[X] = Y$ **Return** Y

Figure 8: Game used in the proof of Theorem 3.

where $\mathrm{Adv}^{\mathrm{ROR}}_{\mathcal{A}',\Pi}(\ell)$ is the advantage of a PPT adversary $\mathcal{A}'$ in the real-or-random (ROR) game for π defined in Fig. 4 of Section 4.3.

$\mathcal{A}'$ is given oracle access to some function O, and its goal is to determine whether the returned string is randomly chosen or generated by E of Π.

In detail: $\mathcal{A}'$ is given input ℓ and access to an oracle O.

1. Initialize $Q=\emptyset$, run $\mathcal{S}(1^\ell)$ to obtain $\{seed_1,...,seed_l,z\}$, choose $\tau \xleftarrow{\$} \{0,1\}^\ell$.
2. Query O on $seed_1,...,seed_l$. O chooses $b^\star \xleftarrow{\$} \{0,1\}$. If $b^\star=1$, O computes $c_{i,1}=\mathsf{E}(r_i,\tau)$, where $r_i \xleftarrow{\$} \{0,1\}^\ell$. If $b^\star=0$, O sets $c_{i,1} \xleftarrow{\$} \{0,1\}^{|\mathsf{E}(r_i,\tau)|}$. O returns $\{c_{i,1}, \forall i \in [1,l]\}$.
3. Choose $c_{i,2} \xleftarrow{\$}, \forall i \in [1,l]$, set $c_i = c_{i,1}||c_{i,2}$ and $c = \{c_1,...,c_l\}$, and send c to $\mathcal{A}$.
4. Whenever $\mathcal{A}$ queries HO on X, choose $Y \xleftarrow{\$} \{0,1\}^\ell$, set $Q[X]=Y$ and return Y. If $seed_i \in Q$, set $\mathsf{flag}=1$.
5. Choose $o \xleftarrow{\$} \{0,1\}$. Output flag if $o=1$ and b' otherwise.

The view of $\mathcal{A}$ when run as a subroutine by $\mathcal{A}'$ in the above procedure is identical with the view of $\mathcal{A}$ in G_5 (when $o=1$) and G_6 (when $o=0$). Therefore,

$$\begin{aligned}
&\Pr[\mathrm{ROR}^{b^\star=1}_{\mathcal{A}',\Pi} \Rightarrow 1 | o=0] - \Pr[\mathrm{ROR}^{b^\star=0}_{\mathcal{A}',\Pi} \Rightarrow 1 | o=0] \\
=&\Pr[G_5 \Rightarrow 1] - \Pr[G_6 \Rightarrow 1], \\
&\Pr[\mathrm{ROR}^{b^\star=1}_{\mathcal{A}',\Pi} \Rightarrow 1 | o=1] - \Pr[\mathrm{ROR}^{b^\star=0}_{\mathcal{A}',\Pi} \Rightarrow 1 | o=1] \\
=&\Pr[\mathsf{flag}=1 \text{ in } G_5] - \Pr[\mathsf{flag}=1 \text{ in } G_6].
\end{aligned}$$

Furthermore, we have

$$\begin{aligned}
&\mathrm{Adv}^{\mathrm{ROR}}_{\mathcal{A}',\Pi}(\ell) \\
=&\frac{1}{2}(\Pr[\mathrm{ROR}^{b^\star=1}_{\mathcal{A}',\Pi} \Rightarrow 1 | o=1] - \Pr[\mathrm{ROR}^{b^\star=1}_{\mathcal{A}',\Pi} \Rightarrow 1 | o=0]) \\
+&\frac{1}{2}(\Pr[\mathrm{ROR}^{b^\star=0}_{\mathcal{A}',\Pi} \Rightarrow 1 | o=1] - \Pr[\mathrm{ROR}^{b^\star=0}_{\mathcal{A}',\Pi} \Rightarrow 1 | o=0]) \\
=&\frac{1}{2}(\Pr[G_5 \Rightarrow 1] + \Pr[\mathsf{flag}=1 \text{ in } G_5]) \\
-&\frac{1}{2}(\Pr[G_6 \Rightarrow 1] + \Pr[\mathsf{flag}=1 \text{ in } G_6]).
\end{aligned}$$

Then, we prove that $\Pr[\mathsf{flag}=1 \text{ in } G_6] \le \frac{ql}{2^\beta}$.

$\mathcal{M}$ is a source with β-entropy α-samples, $\Pr[X=seed_i|z] \le 1/2^\beta$ holds for each $i \in [1,l]$. Therefore,

$$\Pr[\mathsf{flag}=1 \text{ in } G_6] \le ql \cdot \Pr[X=seed_i|z] \le \frac{ql}{2^\beta}.$$

This concludes the proof. □

Theorem 3. *If EpiOracle is IND$-CDA secure for symptom content, it is IND-RTA secure for symptom frequency.*

Proof. We note that in IND-RTA, $d=0$ provides the easiest case for $\mathcal{A}$ to tell whether $b=0$ or $b=1$. Therefore, $\Pr[\text{IND-RTA}_{\mathcal{A}} \Rightarrow 1] \le \Pr[\text{IND-RTA}_{\mathcal{A}} \Rightarrow 1 | d=0]$. Moreover, G_7 shown in Fig. 8 is identical with $\text{IND-RTA}^{d=0,b=1}_{\mathcal{A}}$, then

$$\Pr[\text{IND-RTA}_{\mathcal{A}} \Rightarrow 1 | d=0, b=1] = \Pr[G_7 \Rightarrow 1].$$

In G_8, c_2 is a random string with the same length of c_1, then we have

$$\Pr[G_8 \Rightarrow 1] = \Pr[G_7 \Rightarrow 1] + \mathrm{Adv}^{\text{IND-\$CDA}}_{\mathcal{A}'}(\ell),$$

where $\mathcal{A}'$ is a PPT adversary.

G_9 is identical with G_8, except that c_2 is replaced with a ciphertext of $m_2 \neq m_1$ instead of a random string in G_9, then we have

$$\Pr[G_9 \Rightarrow 1] = \Pr[G_8 \Rightarrow 1] + \mathrm{Adv}^{\text{IND-\$CDA}}_{\mathcal{A}'}(\ell).$$

Actually, G_9 is identical with $\text{IND-RTA}^{d=0,b=0}_{\mathcal{A}}$. Therefore,

$$\begin{aligned}
\mathrm{Adv}^{\text{IND-RTA}}_{\mathcal{A}}(\ell) &\le \Pr[G_9 \Rightarrow 1] - \Pr[G_7 \Rightarrow 1] \\
&= 2 \cdot \mathrm{Adv}^{\text{IND-\$CDA}}_{\mathcal{A}'}(\ell).
\end{aligned}$$

This concludes the proof. □

Security enhancement. DGA is a potential threat towards EpiOracle if an adversary can narrow down the space of symptom lists significantly and try every possible symptom list, he can break the confidentiality of symptom and frequency. We also propose a server-aided scheme [45] to resist such an adversary as follows.

As shown in Fig. 9, to resist DGA, the seeds used to generate the encryption keys are hardened by a key server $\mathcal{KS}$ (which holds a server-side secret key sk), and the interactions between $\mathcal{KS}$ and $\mathcal{H}_i$ are oblivious such that no information about the symptoms would be revealed to $\mathcal{KS}$. Specifically, after generating a symptom list $m_i = m_i[1]||\cdots||m_i[b]$, for each $k \in [1,l]$, a facility $\mathcal{H}_i$ randomly chooses α bytes of m_i, concatenates them as a seed $seed_{i,k}$, requests a signature $\sigma_{i,k}$ on $seed_{i,k}$ under $\mathcal{KS}$'s secret key sk in an oblivious way, derives a key $key_{i,k}$ using $\sigma_{i,k}$, and encrypts the tag tau_i using $key_{i,k}$ to obtain $c_{i,k}$. Finally, p_i contains all indexes of chosen bytes and ciphertexts.

Later, when a healthcare facility $\mathcal{H}_j$ generates a symptom lists m_j, it checks whether m_j is similar to m_i using p_i as follows. For each ciphertext $c_{i,k}$ and the corresponding indexes $ind^{(1)}_{i,k},...,ind^{(\alpha)}_{i,k}$, $k \in [1,l]$ contained in p_i, $\mathcal{H}_j$ computes $seed'_{i,k}$ as $m_j[ind^{(1)}_{i,k}]||\cdots||\,m_j[ind^{(\alpha)}_{i,k}]$, obtains $key'_{i,k}$ following the same steps as that $\mathcal{H}_i$ obtains $key_{i,k}$, and further decrypts $c_{i,k}$ using $key'_{i,k}$. If $key'_{i,k}=key_{i,k}$, $\mathcal{H}_j$ can successfully decrypt $c_{i,k}$ and recover τ_i.

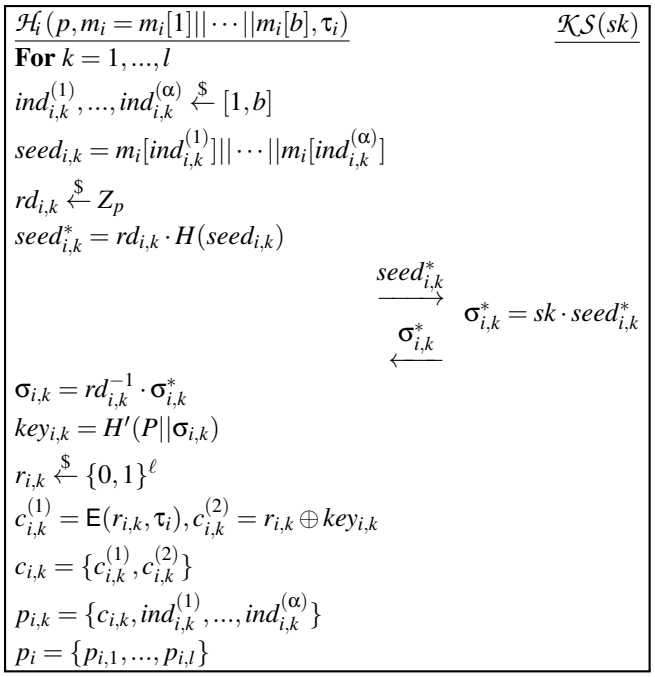


Figure 9: Server-aided helper parameter generation.

By doing so, even if the adversary guesses exact m_i, he can only obtain the valid decryption key in a negligible probability without interacting with $\mathcal{KS}$. Moreover, before requesting a hardened seed from $\mathcal{KS}$, healthcare facilities blind the seed using a fresh randomness. Therefore, $\mathcal{KS}$ cannot obtain any information about the symptom lists.

In this server-aided paradigm, $\mathcal{KS}$ is a single point of failure: once $\mathcal{KS}$ misbehaves, DGA still works. To address the single-point-of-failure problem, the above method can be extended to a multi-servers paradigm, i.e., instead of relying on a single key server $\mathcal{KS}$, a group of key servers $\mathcal{KS}_1, ..., \mathcal{KS}_v$ sharing a server-side secret key in a threshold way are employed to collaboratively harden the seeds. By doing so, DGA is resisted even if several key servers are compromised (the number of compromised key servers does not exceed the threshold). The details can be found in [18, 23, 65] and we do not repeat here to avoid redundancy.

With this enhancement, **EpiOracle** is able to resist DGA. Whereas we have to accept that non-trivial costs in terms of computation and communication would be introduced due to the employment of public-key cryptographic primitives, e.g., oblivious pseudorandom functions [28, 29, 38].

8 Implementation and Evaluation

8.1 Implementation

We implement an EpiOracle prototype in Java with JPBC library [2], and the source code is available at https://anonymous.4open.science/r/EpiOracle-BB20.

In the implementation, we choose [CTR]AES with 256-bit key length as the symmetric-key encryption Π, SHA-256 to implement the hash functions $H, h_1, ..., h_s$, and the data structure BitSet to implement the Bloom Filter $\overrightarrow{BF}$. Moreover, we set α in a length-dependent way, i.e., for a symptom list m_i, the number of positions chosen from a symptom list to generate the encryption keys is computed as $SimRatio \cdot |m_i|$. In the implementation of fuzzy detection, not only the symptom lists themselves but their all possible permutations are compared.

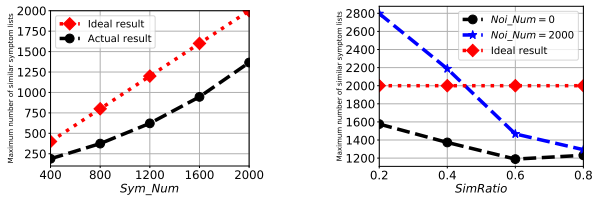
8.2 Evaluation method and setting

We simulate the scenario of the COVID-19 epidemic and conduct experiments on our prototype with various system parameters of EpiOracle for the evaluation in the aspects of accuracy and efficiency. We evaluate the accuracy of EpiOracle in terms of fuzzy detection and insertion count, we also evaluate its efficiency in terms of computation costs and communication costs.

To evaluate the accuracy of the fuzzy detection, we first use a real-world dataset consisting of 200 symptom lists associated with COVID-19. Each list is processed with **FuzDet** and **HelParGen** to generate a tag, and the maximum number of lists with the same tag was calculated and compared with the ideal value, i.e. the total number of lists. In additional, we evaluate the accuracy of fuzzy detection using a larger sample of synthesized lists. We also generated lists comprising different symptoms to simulate "noisy" data that could occur with other diseases and conducted experiments on this dataset in the same manner.

To evaluate the accuracy of insertion count, we generate a set of 32-bit random tags and insert them into the Bloom Filter using the method described in **Increase**. We ensure that one tag, denoted as τ, is inserted the threshold number (denoted as t) of times while others are inserted less than t times. We then count the occurrences of τ based on the Bloom Filter, and compute the explicit threshold of each tag (i.e., the expected number of filled slots of a tag after the threshold number of insertions performed on the tag) using **Theorem 1**, and compare the count with the explicit threshold.

Now we present how we synthesize the symptom lists used for evaluation. According to the statistical proportions of the symptoms reported by WHO [7], we generate 2000 symptom lists associated with COVID-19 and encode them using SNOMED CT. We initialize 2000 empty lists and, for each symptom and its proportion p, randomly insert the symptom and its corresponding code into $p \cdot 2000$ lists. We also generated 2000 noisy lists comprising some common symptoms such as Stomachache (33.4%), Hypothermia (87.9%), Dirty sputum (14.8%), Angina (18.6%), and Genus herpestes (38.1%). All experiments are performed on a laptop equipped with an Intel Core i5 CPU and 16 GB LPDDR4X RAM.



(a) $SimRatio = 0.6$. (b) $Sym\_Num = 2000$.

Figure 10: Results of fuzzy detection.

8.3 Accuracy

Fuzzy detection. We first conducted experiments on the 200 real-world symptom lists with varying values of $(l, SimRatio)$, where l denotes the number of ciphertexts in each helper parameter and $SimRatio$ denotes the ratio of the number of bytes selected from a symptom list for the generation of encryption keys to the length of the corresponding symptom list. The experimental results are presented in Table 2. From the results, we observed that the accuracy of fuzzy detection of EpiOracle is around 70% when $(l, SimRatio) = (10, 0.8)$.

Subsequently, we conduct experiments using the synthesized lists with $l = 1$ and $SimRatio = 0.6$. We vary the number of symptom lists (denoted by Sym_Num) of COVID-19. We also set $l = 10$, $Sym\_Num = 2000$, and vary $SimRatio$ and the number of noisy symptom lists (denoted by Noi_Num). The results are presented in Fig. 10.

The results reveal that the accuracy of EpiOracle decreases with an increase in $SimRatio$ in the absence of noise. However, in practical scenarios, symptom lists of other diseases are also present, and a smaller $SimRatio$ can lead to a higher likelihood of detecting disparate symptom lists as similar. Moreover, the impact of noise on the accuracy of EpiOracle decreases with an increase in $SimRatio$. For instance, when $SimRatio = 0.6$ and 2000 symptom lists of COVID-19 are provided, EpiOracle detects around 1350 similar lists. With the addition of 2000 noisy lists, the number of similar lists detected by EpiOracle increases to about 1450.

$SimRatio$	l	Number of similar lists detected by EpiOracle				
0.6	1	136	109	92	57	75
	5	123	130	86	89	125
	10	97	131	85	123	131
0.7	1	77	93	83	85	142
	5	144	161	142	136	69
	10	146	142	141	141	141
0.8	1	75	64	107	64	78
	5	141	142	138	142	145
	10	140	144	141	141	141

Table 2: Experimental results of fuzzy detection based on 200 real-world symptom lists.

Increment count. The count of a tag is increased by inserting the tag into the Bloom Filter, i.e., randomly choosing an unfilled slot of the tag's item set and setting the slot to 1. The number of slots in an item set, denoted by s, and the number of slots of the Bloom Filter, denoted by L, along with the number of insertions performed on the target tag, denoted by t, and the number of the insertions performed on other tags, denoted by ι, can impact the accuracy of the increment count. To evaluate this impact, we vary the values of (s, L, t, ι), compute the corresponding T, which we present in Table 3, and conduct the following experiments.

We conduct experiments to evaluate the accuracy of increment count by varying the number of insertions performed on a target tag ($300 \leq t \leq 1200$) and on other tags ($600 \leq \iota \leq 2400$), the number of slots in an item set ($2^{11} \leq s \leq 2^{12}$), and the number of slots in the Bloom Filter ($2^{12} \leq L \leq 2^{16}$). The results, presented in Fig. 11, demonstrate that using T as the explicit threshold instead of t can significantly improve the accuracy of increment count. When the total number of insertions exceeds 1500, T and the actual counting result are almost the same. For 3600 insertions, EpiOracle achieves a relatively high accuracy of increment count by setting $L = 2^{16}$ and $s = 2^{12}$, resulting in an 8 KB Bloom Filter size.

8.4 Efficiency

Computation costs. We evaluate the impact of parameters l, α, L, and s on the computation costs incurred by the facility side. We conduct 2000 experiments with different (l, α, L, s)

T \ ι / t	600	1200	1800	2400
300	600	900	1200	1500
600	900	1200	1500	1800
900	1200	1500	1800	2047
1200	1500	1800	2048	2048

(a) $L = 2^{12}, s = 2^{11}$.

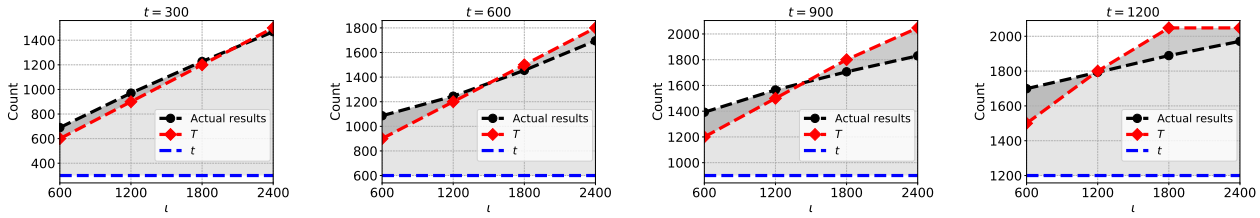
T \ ι / t	600	1200	1800	2400
300	375	450	506	575
600	719	781	818	879
900	975	1050	1125	1200
1200	1275	1350	1425	1500

(b) $L = 2^{14}, s = 2^{11}$.

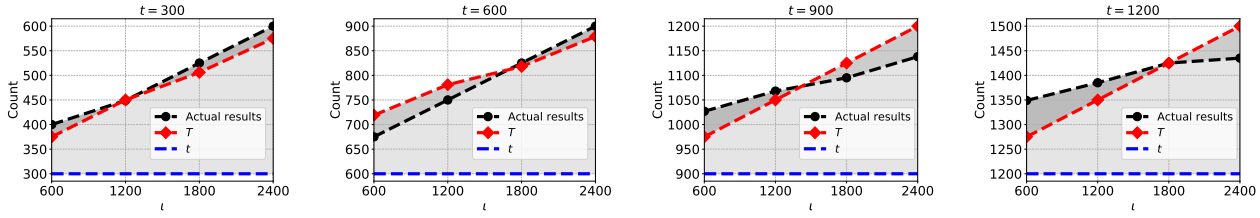
T \ ι / t	600	1200	1800	2400
300	344	365	413	450
600	638	675	712	750
900	938	975	1013	1050
1200	1238	1275	1313	1350

(c) $L = 2^{16}, s = 2^{12}$.

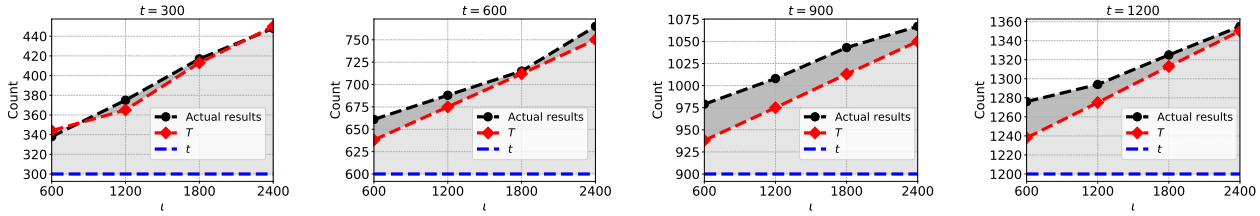
Table 3: Explicit threshold (T).



(a) $L = 2^{12}, s = 2^{11}$.

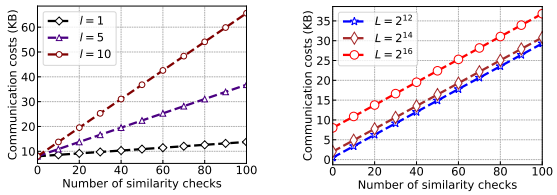


(b) $L = 2^{14}, s = 2^{11}$.



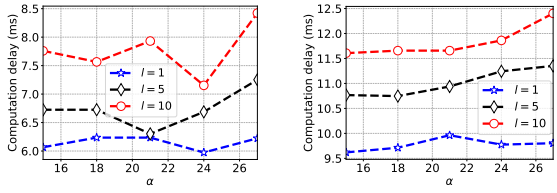
(c) $L = 2^{16}, s = 2^{12}$.

Figure 11: Evaluation results of increment count.



(a) $L = 2^{16}$. (b) $l = 5$.

Figure 13: Communication costs between a facility and cloud.



(a) $L = 2^{12}, s = 2^{11}$. (b) $L = 2^{16}, s = 2^{12}$.

Figure 12: Computation delay on the facility side.

configurations and present the average computation delay in Fig. 12. Notably, when $(l, \alpha, L, s) = (10, 27, 2^{16}, 2^{12})$, the average computation delay is around 12.4 ms, which is considered acceptable for the facilities.

Communication costs. In terms of communication costs, we measure the costs incurred by obtaining the Bloom Filter and outsourced helper parameters from the cloud server, and sending helper parameters and hash values of tags to the cloud server for count increase. We set $L = 2^{20}$ (resulting in a 32 KB Bloom Filter), and conduct experiments with different l and L. The results are presented in Fig. 13. We observe that when $L = 2^{21}$ and 100 similarity checks are performed, the communication costs on the facility side are less than 40 KB, which is highly efficient.

9 Conclusion

In this paper, we have analyzed the security of syndrome-based early epidemic warning systems and formalized two security notions: symptom confidentiality and frequency confidentiality. We have developed EpiOracle, a cross-facility syndrome-based early unknown epidemic warning scheme with privacy preservation. We have formally proved that EpiOracle is secure in the random oracle model. We have implemented an EpiOracle prototype and evaluated its performance. The evaluation results demonstrate the practicality and efficiency of EpiOracle. For future work, we will investigate how to detect similar symptom lists using other similarity measurements such as edit distance and set difference.

References

[1] Surveillance for a count data time series using the EARS C1, C2 or C3 method and its extensions. https://rdrr.io/cran/surveillance/man/earsC.html.

[2] The java pairing based cryptography library (JPBC). http://gas.dia.unisa.it/projects/jpbc/#.Yg9TnC-KFqs, 2018.

[3] 2009 H1N1 Pandemic. https://www.cdc.gov/flu/pandemic-resources/2009-h1n1-pandemic.html, 2019.

[4] Google's secret cache of medical data includes names and full details of millions – whistleblower. https://www.theguardian.com/technology/2019/nov/12/google-medical-data-project-nightingale-secret-transfer-us-health-information, 2019.

[5] Coronavirus: The psychology of panic buying. https://www.bbc.com/worklife/article/20200304-coronavirus-covid-19-update-why-people-are-stockpiling, 2020.

[6] Guide for the biosense platform. https://www.cdc.gov/nssp/biosense/onboarding-guide/pdf/New-Site-OG-508.pdf, 2020.

[7] Report of the WHO-China joint mission on coronavirus disease 2019 (COVID-19). Technical report, World Health Organization (WHO), 2 2020.

[8] 'it's crazy': Panic buying forces stores to limit purchases of toilet paper and masks. https://www.cnn.com/2020/03/06/business/coronavirus-global-panic-buying-toilet-paper/index.html, 2020.

[9] China covid: 'panic-buying' and shortages as restrictions are eased. https://www.bbc.com/news/world-asia-china-63976197, 2022.

[10] ICD-11. https://icd.who.int, 2022.

[11] Nearly one billion people in china had their personal data leaked, and it's been online for more than a year. https://edition.cnn.com/2022/07/05/china/china-billion-people-data-leak-intl-hnk/index.html, 2022.

[12] Medical dictionary for regulatory activities. https://www.meddra.org, 2023.

[13] National outbreak report system. https://www.cdc.gov/nors/index.html., 2023.

[14] National syndromic surveillance program (NSSP). https://www.cdc.gov/nssp/how-sys.html., 2023.

[15] ProMED-mail. https://promedmail.org, 2023.

[16] Snomed ct. https://www.snomed.org, 2023.

[17] WHO coronavirus (covid-19) dashboard. https://covid19.who.int, 2023.

[18] Shashank Agrawal, Peihan Miao, Payman Mohassel, and Pratyay Mukherjee. PASTA: password-based threshold authentication. In *Proc. CCS*, pages 2042–2059, 2018.

[19] Adil Ahmad, Juhee Kim, Jaebaek Seo, Insik Shin, Pedro Fonseca, and Byoungyoung Lee. Chancel: Efficient multi-client isolation under adversarial programs. In *Proc. NDSS*, 2021.

[20] Emily Alsentzer, Sarah-Blythe Ballard, Joan Neyra, Delphis M Vera, Victor B Osorio, Jose Quispe, David L Blazes, and Luis Loayza. Assessing 3 outbreak detection algorithms in an electronic syndromic surveillance system in a resource-limited setting. *Emerging Infectious Diseases*, 26(9):2196, 2020.

[21] Gilad Asharov, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Ariel Nof, Benny Pinkas, Katsumi Takahashi, and Junichi Tomida. Efficient secure three-party sorting with applications to data analysis and heavy hitters. In *Proc. CCS*, pages 125–138, 2022.

[22] Borja Balle and Yu-Xiang Wang. Improving the gaussian mechanism for differential privacy: Analytical calibration and optimal denoising. In *Proc. ICML*, pages 394–403, 2018.

[23] Carsten Baum, Tore Frederiksen, Julia Hesse, Anja Lehmann, and Avishay Yanai. Pesto: proactively secure distributed single sign-on, or how to trust a hacked server. In *Proc. EuroS&P*, pages 587–606, 2020.

[24] Mihir Bellare, Sriram Keelveedhi, and Thomas Ristenpart. Message-locked encryption and secure deduplication. In *Proc. EUROCRYPT*, pages 296–312, 2013.

[25] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In *Proc. EUROCRYPT*, pages 409–426, 2006.

[26] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[27] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Lightweight techniques for private heavy hitters. In *Proc. S&P*, pages 762–776, 2021.

[28] Dan Boneh, Dmitry Kogan, and Katharine Woo. Oblivious pseudorandom functions from isogenies. In *Proc. ASIACRYPT*, pages 520–550, 2020.

[29] Dan Boneh, Hart William Montgomery, and Ananth Raghunathan. Algebraic pseudorandom functions with improved efficiency from the augmented cascade. In *Proc. CCS*, pages 131–140, 2010.

[30] Ran Canetti, Benjamin Fuller, Omer Paneth, Leonid Reyzin, and Adam Smith. Reusable fuzzy extractors for low-entropy distributions. *Journal of Cryptology*, 34(1):1–33, 2021.

[31] Sylvain Chatel, Apostolos Pyrgelis, Juan Ramón Troncoso-Pastoriza, and Jean-Pierre Hubaux. Privacy and integrity preserving computations with CRISP. In *Proc. USENIX Security*, pages 2111–2128, 2021.

[32] Long Chen, Ya-Nan Li, Qiang Tang, and Moti Yung. End-to-same-end encryption: modularly augmenting an app with an efficient, portable, and blind cloud storage. In *Proc. USENIX Security*, pages 2353–2370, 2022.

[33] Nanshan Chen, Min Zhou, Xuan Dong, Jieming Qu, Fengyun Gong, Yang Han, Yang Qiu, Jingli Wang, Ying Liu, Yuan Wei, et al. Epidemiological and clinical characteristics of 99 cases of 2019 novel coronavirus pneumonia in Wuhan, China: a descriptive study. *The Lancet*, 395(10223):507–513, 2020.

[34] Tobias Cloosters, Michael Rodler, and Lucas Davi. Teerex: discovery and exploitation of memory corruption vulnerabilities in sgx enclaves. In *Proc. USENIX Security*, pages 841–858, 2020.

[35] Anders Dalskov, Daniel Escudero, and Marcel Keller. Fantastic four:{Honest-Majority}{Four-Party} secure computation with malicious security. In *Proc. USENIX Security*, pages 2183–2200, 2021.

[36] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *Proc. TCC*, pages 285–304, 2006.

[37] Centers for Disease Control and Prevention. Unexplained respiratory disease outbreak working group activities-worldwide, march 2007-september 2011. *MMWR. Morbidity and Mortality Weekly Report*, 61(26):480–483, 2012.

[38] Michael J Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In *Proc. PKC*, pages 303–324, 2005.

[39] Deborah W Gould, David Walker, and Paula W Yoon. The evolution of biosense: lessons learned and future directions. *Public Health Reports*, 132(1_suppl):7S–11S, 2017.

[40] Marcus Hähnel, Weidong Cui, and Marcus Peinado. High-resolution side channels for untrusted operating systems. In *Proc. USENIX ATC*, pages 299–312, 2017.

[41] Roger A Hallman, Kim Laine, Wei Dai, Nicolas Gama, Alex J Malozemoff, Yuriy Polyakov, and Sergiu Carpov. Building applications with homomorphic encryption. In *Proce. CCS*, pages 2160–2162, 2018.

[42] Tyler Hunt, Zhiting Zhu, Yuanzhong Xu, Simon Peter, and Emmett Witchel. Ryoan: A distributed sandbox for untrusted computation on secret data. In *Proc. OSDI*, pages 533–549, 2016.

[43] Lori Hutwagner, William Thompson, G Matthew Seeman, and Tracee Treadwell. The bioterrorism preparedness and response early aberration reporting system (ears). *Journal of Urban Health*, 80:i89–i96, 2003.

[44] Burt Kaliski. PKCS# 5: password-based cryptography specification version 2.0. Technical report, 2000.

[45] Sriram Keelveedhi, Mihir Bellare, and Thomas Ristenpart. DupLESS:server-aided encryption for deduplicated storage. In *Proc. USENIX Security)*, pages 179–194, 2013.

[46] Mathias Lecuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. Certified robustness to adversarial examples with differential privacy. In *Proc. S&P*, pages 656–672, 2019.

[47] JaeHyuk Lee, Jinsoo Jang, Yeongjin Jang, Nohyun Kwak, Yeseul Choi, Changho Choi, Taesoo Kim, and Brent Byunghoon Kang. Hacking in darkness: Return-oriented programming against secure enclaves. pages 523–539, 2017.

[48] Linsheng Liu, Daniel S Roche, Austin Theriault, and Arkady Yerukhimovich. Fighting fake news in encrypted messaging with the fuzzy anonymous complaint tally system (FACTS). In *Proc. NDSS*, 2022.

[49] Eleftheria Makri, Dragos Rotaru, Frederik Vercauteren, and Sameer Wagh. Rabbit: Efficient comparison for secure multi-party computation. In *Proc. FC*, pages 249–270, 2021.

[50] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R Savagaonkar. Innovative instructions and software model for isolated execution. *Hasp@ isca*, 10(1), 2013.

[51] Faisal S Minhaj, Yasmin P Ogale, Florence Whitehill, Jordan Schultz, Mary Foote, Whitni Davidson, Christine M Hughes, Kimberly Wilkins, Laura Bachmann, Ryan Chatelain, et al. Monkeypox outbreak—nine states, may 2022. *Morbidity and Mortality Weekly Report*, 71(23):764, 2022.

[52] Mohammad Norouzi, David J Fleet, and Russ R Salakhutdinov. Hamming distance metric learning. *Advances in Neural Information Processing Systems*, 25, 2012.

[53] Sarvar Patel, Giuseppe Persiano, Kevin Yeo, and Moti Yung. Mitigating leakage in secure cloud-hosted data structures: Volume-hiding for multi-maps via hashing. In *Proc. CCS*, pages 79–93, 2019.

[54] Edo Roth, Daniel Noble, Brett Hemenway Falk, and Andreas Haeberlen. Honeycrisp: large-scale differentially private aggregation without a trusted core. In *Proc. SOSP*, pages 196–210, 2019.

[55] Edo Roth, Hengchu Zhang, Andreas Haeberlen, and Benjamin C Pierce. Orchard: Differentially private analytics at scale. In *Proc. OSDI*, pages 1065–1081, 2020.

[56] Felix Schuster, Manuel Costa, Cédric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, and Mark Russinovich. Vc3: Trustworthy data analytics in the cloud using sgx. In *Proc. S&P*, pages 38–54, 2015.

[57] Alex Skvortsov and Branko Ristic. Monitoring and prediction of an epidemic outbreak using syndromic observations. *Mathematical Biosciences*, 240(1):12–19, 2012.

[58] Flavio Toffalini, Mariano Graziano, Mauro Conti, and Jianying Zhou. Snakegx: A sneaky attack against sgx enclaves. In *Proc. ACNS*, pages 333–362, 2021.

[59] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the keys to the intel sgx kingdom with transient out-of-order execution. In *Proc. USENIX Security*, pages 991–1008, 2018.

[60] Longde Wang, Yu Wang, Shuigao Jin, Zunyou Wu, Daniel P Chin, Jeffrey P Koplan, and Mary Elizabeth Wilson. Emergence and control of infectious diseases in china. *The Lancet*, 372(9649):1598–1605, 2008.

[61] Weier Wang, Jianming Tang, and Fangqiang Wei. Updated understanding of the outbreak of 2019 novel coronavirus (2019-nCoV) in wuhan, china. *Journal of Medical Virology*, 92(4):441–447, 2020.

[62] Zunyou Wu and Jennifer M McGoogan. Characteristics of and important lessons from the coronavirus disease 2019 (covid-19) outbreak in china: summary of a report of 72 314 cases from the chinese center for disease control and prevention. *JAMA*, 323(13):1239–1242, 2020.

[63] Qingqing Ye, Haibo Hu, Xiaofeng Meng, and Huadi Zheng. Privkv: key-value data collection with local differential privacy. In *Proc. S&P*, pages 317–331, 2019.

[64] Chengliang Zhang, Suyi Li, Junzhe Xia, Wei Wang, Feng Yan, and Yang Liu. Batchcrypt: Efficient homomorphic encryption for cross-silo federated learning. In *Proc. USENIX ATC*, pages 493–506, 2020.

[65] Yuan Zhang, Chunxiang Xu, Nan Cheng, and Xuemin Shen. Secure password-protected encryption key for deduplicated cloud storage systems. *IEEE Transactions on Dependable and Secure Computing*, 19(4):2789–2806, 2022.

[66] Mingxun Zhou, Tianhao Wang, TH Hubert Chan, Giulia Fanti, and Elaine Shi. Locally differentially private sparse vector aggregation. In *Proc. S&P*, pages 422–439, 2022.

A Preliminaries

Bloom filter. A Bloom Filter is essentially a bit vector initialized to 0 [26]. After inserting a set of elements into a Bloom Filter, it is efficient to check whether an element is a member of the set using the Bloom Filter approximately.

Fix a Bloom Filter $\overrightarrow{v}$ with randomly selected k independent hash functions $\{h_i\}_{i=1}^{k}$, given a set S, an element $a \in S$ is inserted into $\overrightarrow{v}$ by setting all slots in a's item (i.e., the $h_i(a)$-th component of $\overrightarrow{v}$ for each $i \in [1,k]$) to 1.

After inserting all elements in S into $\overrightarrow{v}$, the membership of an element a can be verified by checking whether $\overrightarrow{v}[h_i(a)] = 1, \forall i \in [1,k]$. If not, $a \notin S$. Otherwise, there are two cases: (1) $a \in S$, or (2) a false positive occurs, i.e., $a \notin S$, and all slots in a's item set are set to 1 when inserting other elements.

The variant used in this paper. Fix a Bloom Filter $\overrightarrow{v}$ with randomly selected k independent hash functions $\{h_i\}_{i=1}^{k}$, the tally of an element a is increased by 1 via setting a random slot in a's item where the bit is 0 (i.e., the $h_i(a)$-th component of $\overrightarrow{v}$ for random $i \in [1,k]$ and $\overrightarrow{v}[h_i(a)] = 0$) to 1.

After being increased multiple times, a can be tallied by computing the number of slots set to 1 in a's item. Specifically, let num denote the tally of a and initialized to 0, for each $i \in [1,k]$, if $\overrightarrow{v}[h_i(a)] = 1$, $num = num + 1$.

Hamming distance. Given two strings s_1, s_2 of length n over some arbitrary alphabet Z, the Hamming distance between s_1 and s_2 represents the number of positions where are different [52], which can be mathematically denoted by $dis(s_1, s_2) = |D|$, where $D = \{x | s_1[x] \neq s_2[x], \forall x \in [1,n]\}$.

Min-entropy and high-entropy samples. Given a variable $X = X[1]||\cdots||X[n]$ of length n over some arbitrary alphabet Z and a particular outcome x, its min-entropy is

$$\mathrm{H}_\infty(X) = -\log(\max_x \Pr[X = x]),$$

the average min-entropy of X given another variable Y is

$$\tilde{\mathrm{H}}_\infty(X|Y) = -\log(\mathbb{E}_{y\in Y} \max_x \Pr[X = x|Y = y]),$$

and for some parameters α, β, and uniformly chosen indexes $\{1 \le i_j \le n\}_{j=1}^{\alpha}$, X is a variable with β-entropy α-samples if

$$\tilde{\mathrm{H}}_\infty(X_{i_1}, X_{i_2}, \cdots X_{i_\alpha}|i_1, i_2, ..., i_\alpha) \ge \beta,$$

which is defined in [30].

B Artifact

Abstract

Our artifact consists of an EpiOracle prototype and a database storing the data set used in our experiments. EpiOracle is a syndrome-based early warning system for unknown epidemics. It supports fuzzy detection over the ciphertexts of the symptom lists and the statistics on the frequency of each list as well as its similar ones. The data set is generated according to the symptom information published by WHO.

Scope

Our artifact can be used to prove the correctness and the feasibility of EpiOracle. It can also be used to evaluate the performance of EpiOracle. Specifically, it demonstrates that EpiOracle can be deployed in practice and function well. It can be used to evaluate the computation and communication costs, the accuracy of fuzzy detection and the increment count. It can also be used to validate the evaluation results presented in Section 8.

Content

The artifact comprises the following sub-directories:

- ./EpiOracle, which contains the sourcecode of the EpiOracle prototype.
- ./DataSet, which contains the data set used in the experiments detailed in Section 8.
- Additionally, a README file is included to introduce the build instructions and usage.

Requirements

We developed and evaluated our artifact on a laptop with macOS Monterey 12.5.1, an Intel Core i5 CPU, and 16 GB LPDDR4X of RAM. The prototype is implemented in JAVA with the JPBC library. Moreover, to run the prototype correctly, some basic packages including jna, jpbc-api, jpbc-benchmark, jpbc-crypto, jpbc-mm, jpbc-pbc, jpbc-plaf, c3p0, commons-codec, and mysql-connector-java are required. The versions of these tools are detailed in the README file (https://anonymous.4open.science/r/EpiOracle-BB20/README.md).