EpiOracle: Privacy-Preserving Cross-Facility Early Warning for Unknown Epidemics

Shiyu Li
University of Electronic Science and Technology of China
shiyu@email.virginia.edu

Yuan Zhang*
University of Electronic Science and Technology of China
zhangyuan@uestc.edu.cn

Yaqing Song
University of Electronic Science and Technology of China
YaqingS@163.com

Fan Wu
Central South University
wfwu-fan@csu.edu.cn

Feng Lyu
Central South University
fenglyu@csu.edu.cn

Kan Yang
University of Memphis
kan.yang@memphis.edu

Qiang Tang
The University of Sydney
qiang.tang@sydney.edu.au

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.

Keywords

eHealth systems, early epidemic warning, privacy preservation

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 [16]. 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) [12] and China Infectious Diseases Automated-Alert and Response System (CIDARS) [69]. Such *disease*-based systems have accurately warned the outbreaks of well-studied diseases (e.g., monkeypox [57]). 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) [13, 42] and ProMED-mail [14]. 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 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 [66]. The syndrome-based systems gain a significant advantage in early unknown epidemic warning over the disease-based ones as 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

This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license visit https://creativecommons.org/licenses/by/4.0/ or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.
Proceedings on Privacy Enhancing Technologies 2025(1), 361–378
© 2025 Copyright held by the owner/author(s).
https://doi.org/10.56553/popets-2025-0020

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.

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

- *Fuzzy detection.* 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, 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.

However, existing techniques, including generic multi-party computation (MPC), local differential privacy, private heavy hitters, and trusted hardware, fail to achieve the aforementioned goals simultaneously. We will detail an analysis on them in Section 2.

In this work, we take a step towards privacy-preserving early warning for unknown epidemics by constructing EpiOracle, where cross-facility statistics on symptom frequencies are compiled while symptom contents and frequencies are not leaked to any unrelated party. *To achieve fuzzy detection*, similar symptom lists are detected and mapped to one same tag, and the frequency statistics are compiled by counting the tags. *To prevent symptoms from leakage*, EpiOracle enables the fuzzy detection on the ciphertexts of symptom lists. *To prevent frequencies from leakage*, a counting Bloom Filter is deployed to maintain the counters, which ensures that the status of each tag's counter is kept private from the cloud and all facilities that have not generated the corresponding lists (or its similar ones). Moreover, EpiOracle solely relies on symmetric-key cryptographic primitives, making it highly efficient. We summarize a comparison between the aforementioned techniques and EpiOracle 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[1] 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.

Roadmap. The remainder of this paper is organized as follows. We review the related works in Section 2 and present preliminaries in Section 3. We overview EpiOracle in Section 4 and present its framework in Section 5. We provide the construction of EpiOracle in Section 6, discuss limitations and practical considerations in Section 7, give formal proofs of the security in Section 8, and present the implementation details and evaluation results in Section 9. Finally, we draw the conclusions and outlook the future work in Section 10.

[1]We have obtained consent from the healthcare facility to use the real-world 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. [70] leverages real-world data while ensuring the utmost respect for privacy and ethical standards) to release experimental results, ensuring that no patient privacy would be leaked.

Table 1: Comparisons between EpiOracle and partial solutions.

	Fuzzy detection without symptom leakage	No frequency leakage	Symmetric-key cryptographic primitives only
Generic multi-party computation [46, 60]	Yes	Yes	No
Local differential privacy [47, 73]	No	No	-[1]
Private heavy hitters [21, 27]	No	No	Yes
Trusted hardware [18, 48, 64]	Yes	Yes	-[2]
EpiOracle	Yes	Yes	Yes

[1] Instead of cryptographic primitives, differential privacy is based on mathematical operations and statistical algorithms such as adding Laplace noise.
[2] In hardware-based solutions, the sole cryptographic operations required are those executed to establish secure channels between the trusted hardware and its users.

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.

Typical disease-based early warning systems include NORS [12] and CIDARS [69]. 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 [39]. 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 [13], a syndrome-based early warning system. NSSP employs a cloud server, BioSense [42], 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 [16]. 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 [13] and ProMED-mail [14] 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 [52, 61]. There exists a potential risk of the cloud server attempting to access and exploit sensitive symptom contents for personal gain [32]. Furthermore, the frequencies themselves hold significant value [33], 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.

Generic multi-party computation. Generic multi-party computation (MPC) could be employed to privately monitor the frequencies of symptom lists, where each symptom list is taken as input and compared with other ones for fuzzy detection. In this approach, public-key cryptographic operations, such as homomorphic encryption and oblivious transfer, are intensively utilized [45, 50, 60]. Moreover, the computational overhead grows quadratically with the number of lists and (at least) linearly with the bitlength of each list. For example, a potential solution is to compute Hamming distance between each pair of lists. In [46], a garbled-circuit-based approach for computing Hamming distance is proposed, where given two m-bit strings, $(m \cdot \log m)/2$ non-XOR gates are needed for the distance computation. Therefore, a practical challenge arises due to potential computational costs, especially when the number of lists or the length of each list is large.

Local differential privacy. Previous research has explored the use of local differential privacy to analyze private data [22, 62, 63, 72, 73], which could be adopted in early epidemic warning. Specifically, each healthcare facility individually collects its own frequencies of each symptom list, introduces some noise to the frequencies and then uploads the perturbed data to a central aggregator for aggregation. 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. 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.

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 [56]) for frequencies statistics, where the frequencies of all symptom lists are computed in a secure enclave [18, 48, 64]. 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 it vulnerable to emerging attacks such as side-channel attacks [43, 53, 67, 68]. 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 Preliminaries

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 $F(key, m)$ the evaluation of a keyed function F with the key key and the input m, by $r \xleftarrow{\$} S$ randomly choosing an 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.,

$$\binom{n}{m} = m!/(n!(m-n)!).$$

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 [59], 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 $\{1 \leq i_j \leq n\}_{j=1}^{\alpha}$, X is a variable with β-entropy α-samples if

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

which is defined in [30].

Symptom encoding. Currently, there are some encoding standards such as SNOMED CT [15] 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 unproductive 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 [34]. 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. This will be further discussed in Section 7

4 Overview of EpiOracle

4.1 Warm-up

We introduce a basic scheme to show how early warning is achieved in existing syndrome-based systems (e.g., NSSP [13] and ProMED-mail [14]). As shown in Figure 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., the 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.

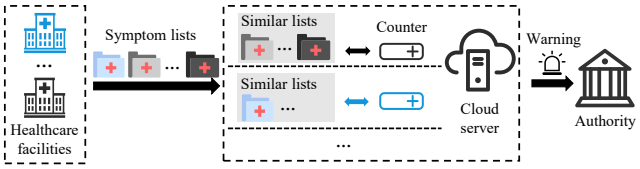


Figure 1: System model of the basic scheme.

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 the introduction, the symptom contents and frequencies should not be leaked to the cloud server.

4.2 Technical overview

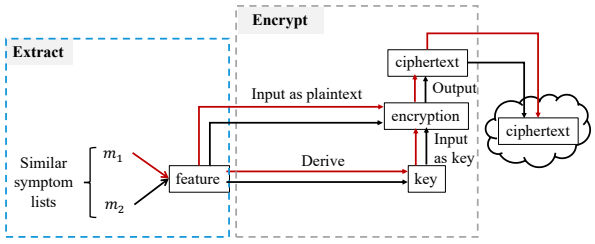
In this paper, we target to construct a practical cross-facility early warning scheme for unknown epidemics while achieving symptom and frequency confidentiality. We overview our solution focusing on the challenges addressed by it.

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 (we stress that any CPA-secure encryption scheme would be randomized), 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.

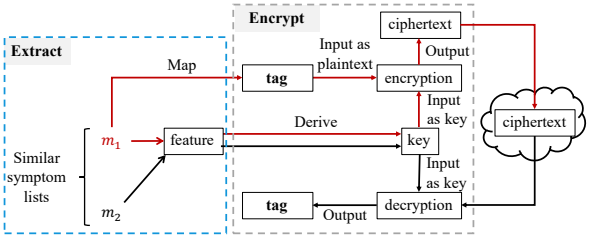
Detecting same plaintexts over encrypted symptom lists. Inspired by encrypted data deduplication, we may utilize message-locked-encryption (MLE) [24], where the encryption/decryption key is derived from the plaintext itself, to protect the symptom contents against leakage while supporting the statistics 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 symptom 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 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 Figure 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



(a) Ensuring symptom confidentiality.



(b) Ensuring symptom and frequency confidentiality.

Figure 2: Extract-then-encrypt mechanism.

can maintain a counter for each ciphertext to monitor the frequencies of (similar) lists.

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 Figure 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 [54]. 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 facility randomly fills

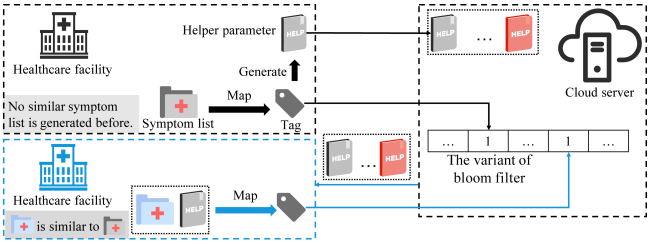


Figure 3: System model of EpiOracle.

a slot of the Bloom Filter in the corresponding tag's item set. The detailed elaboration of the variant of Bloom Filter is provided in Section 3. 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, which uses only symmetric-key primitives to achieve:

- fuzzy detection with symptom confidentiality by enabling each healthcare facility to selectively share information about its symptom lists to certain counterparts without direct interactions,
- frequency statistic with frequency confidentiality by enabling the cloud to update counters of lists without knowing the counts, which also avoids the need for costly tools such as ORAM.

5 Framework of EpiOracle

5.1 System model and threat model

As shown in Figure 3, EpiOracle involves two entities: a cloud server and healthcare facilities.

- Cloud server. The cloud server maintains helper parameters for fuzzy detection. It also maintains a Bloom Filter that records the frequencies of all symptom lists. The cloud server is curious about the content of the symptom lists as well as their frequencies. It may compromise some healthcare facilities for these information.
- Healthcare facilities. When a healthcare facility generates a symptom list, it 1) performs fuzzy detection on its own list and others with the aid of helper parameters maintained by the cloud server; 2) maps the list to the same tag as similar ones and generates a helper parameter using the tag; 3) outsources the helper parameter to the cloud server and increases the frequency of the list by inserting the tag into the Bloom Filter. The facility can also compute the frequency of a list based on the information stored in the Bloom Filter. We assume that the healthcare facilities are semi-honest. They will not input false symptom lists into the scheme to disturb the frequencies. However, they may be compromised by the cloud server and leak the information they hold.

5.2 Syntax

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

$\langle sp, \overrightarrow{BF} \rangle \leftarrow$ **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 detections. 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.

$\langle \overrightarrow{BF'} \rangle \leftarrow$ **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.

5.3 Security definitions

We present security definitions of symptom and frequency confidentiality 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, where 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 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 list and further decrypting the ciphertext contained in the helper parameter.

$\underline{\text{IND\$-CDA}_{\mathcal{A},\mathcal{M}}(\ell)}$
$1: sp \leftarrow \textbf{Setup}(1^\ell)$
$2: m \xleftarrow{\$} \mathcal{M}$
$3: \tau \xleftarrow{\$} \{0,1\}^\ell$
$4: b \xleftarrow{\$} \{0,1\}$
$5: p_1 := (c_1, z) \leftarrow \textbf{HelParGen}(m, \tau)$
$6: c_0 \xleftarrow{\$} \{0,1\}^{|c_1|}$
$7: b' \leftarrow \mathcal{A}(sp, c_b, z)$
$8:$ **Return** $(b = b')$

$\underline{\text{ROR}_{\mathcal{A}',\Pi}(\ell)}$
$1: m \leftarrow \mathcal{A}'^{\mathsf{E}}(1^\ell)$
$2: b \xleftarrow{\$} \{0,1\}$
$3:$ **If** $b = 1$
$4:$ **For** $i = 1, ..., l$
$5:$ $k_i \leftarrow \text{Gen}(1^\ell)$
$6:$ $c_i \leftarrow \mathsf{E}(k_i, m)$
$7:$ **Else**
$8:$ **For** $i = 1, ..., l$
$9:$ $c_i \xleftarrow{\$} \{0,1\}^{|\mathsf{E}(k_i,m)|}$
$10: b' \leftarrow \mathcal{A}'(1^\ell, \{c_i\}_{i=1}^l)$
$11:$ **Return** $(b = b')$

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

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

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

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

where $Adv^{\text{ROR}}_{\mathcal{A}',\Pi}(\ell)$ is the advantage of a PPT adversary $\mathcal{A}'$ in the real-or-random game of a symmetric-key encryption $\Pi = (Gen, E, D)$ defined in Figure 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[2].

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.

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.

[2]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 in Appendix C.

$\underline{\text{IND-RTA}_{\mathcal{A},\mathcal{M}}(\ell)}$
$1: sp \leftarrow \textbf{Setup}(1^\ell)$
$2: m_1 \xleftarrow{\$} \mathcal{M}, \tau_1 \xleftarrow{\$} \{0,1\}^*$
$3: p_1 \leftarrow \textbf{HelParGen}(m_1, \tau_1)$
$4: b \xleftarrow{\$} \{0,1\}$
$5:$ **If** $b = 1$
$6:$ $m_2 \xleftarrow{\$} \mathcal{M}$, s.t. $dis(m_1, m_2) \leq d, \tau_2 := \tau_1$
$7:$ **If** $b = 0$
$8:$ $m_2 \xleftarrow{\$} \mathcal{M}$, s.t. $dis(m_1, m_2) > d, |m_2| = |m_1|$
$9:$ $\tau_2 \xleftarrow{\$} \{0,1\}^*$
$10: \overrightarrow{BF}' \leftarrow \textbf{Increase}(\tau_2, \overrightarrow{BF})$
$11: p_2 \leftarrow \textbf{HelParGen}(m_2, \tau_2)$
$12: b' \leftarrow \mathcal{A}(sp, p_1, p_2, \overrightarrow{BF}')$
$13:$ **Return** $(b = b')$

Figure 5: Game defining IND-RTA.

We illustrate this property in the game shown in Figure 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 $\mathcal{A}'$, any β-entropy l-samples source $\mathcal{M}$*

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

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.

6 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. We detail the construction of EpiOracle below, we also illustrate the workflow of EpiOracle and its instantiation in Figure 6.

Assumption. We assume that there is a secure authentication mechanism behind EpiOracle to establish a secure channel between $\mathcal{H}_i$ and $\mathcal{CS}$ for each $i \in [1, N]$. There have been various standards for authentication [38, 40, 58]. According to these standards, various mechanisms [19, 44, 65] can be *directly* integrated, and we would not explicitly describe it for the sake of brevity. As such, we assume that the interactions between the healthcare facility and the cloud server are authenticated and cannot be tampered with.

Setup. With the security parameter ℓ, the system parameter $sp = \{L, t, l, \alpha, P, h_1, ..., h_s, H, \Pi\}$ is determined and $\overrightarrow{BF}$ is initialized, where $\overrightarrow{BF}$ is an L-bit vector initialized as an empty Bloom Filter, t is the warning threshold with $t < L$, l is the number of rounds in helper parameter generation, α is the number of positions sampled

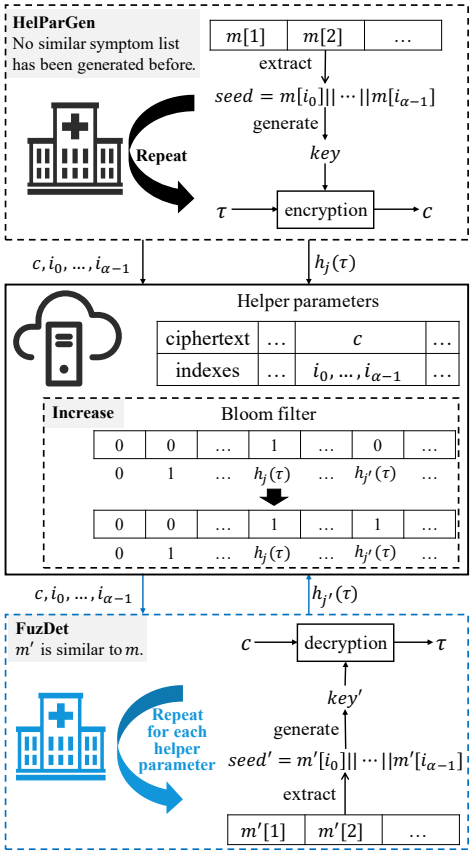


Figure 6: Workflow and an instantiation of EpiOracle.

from a symptom list, $P \in \{0,1\}^\ell$ is a random value, $h_1, ..., h_s : \{0,1\}^* \to \{0,1\}^L, H : \{0,1\}^* \to \{0,1\}^\ell$ are hash functions ($s < t$), $\Pi = (\mathsf{Gen}, \mathsf{E}, \mathsf{D})$ is a symmetric-key encryption scheme with wrong-key detection[3], $\mathsf{E}(K, M)$ 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 containing multiple lists and show how $\mathcal{H}_i$ checks whether m_i is similar to $m_{i'}$ that has been generated before. The smaller the Hamming distance between two lists, the more likely they are detected as similar, but the Hamming distance does not appear explicitly in the scheme.

- For existing help parameters p_j of $m_j (\forall j \in [1, i])$ publicly 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.

[3]Wrong-key detection can be achieved through a simple transformation. Given a traditional symmetric-key encryption scheme $(\mathsf{Gen}', \mathsf{E}', \mathsf{D}')$, $\Pi = (\mathsf{Gen}, \mathsf{E}, \mathsf{D})$ can be constructed as follows: $\mathsf{Gen} := \mathsf{Gen}'$, $\mathsf{E}(\mathsf{K}, \mathsf{M}) := \begin{cases} R \xleftarrow{\$} \{0,1\}^\ell \\ \text{output } C = (R, \mathsf{E}'(K, M||R)) \end{cases}$, $\mathsf{D}(\mathsf{K}, \mathsf{C}=(\mathsf{R}, \mathsf{CT})) := \begin{cases} M'||R' = \mathsf{D}'(K, CT) \\ \text{output } M' \text{ if } R = R' \text{ and} \perp \text{otherwise} \end{cases}$, where $\perp$ is an error symbol to indicate that decryption fails. This transformation is detailed by Lemma 1 in the full version of [31].

Algorithm 1: Fuzzy Detection

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: Helper Parameter Generation

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**
  $ind_{i,k}^{(1)}, ..., ind_{i,k}^{(\alpha)} \xleftarrow{\$} [1, b]$;
  $seed_{i,k} = m_i[ind_{i,k}^{(1)}]|| \cdots ||m_i[ind_{i,k}^{(\alpha)}]$;
  // randomly choose α bytes from m_i
  $key_{i,k} = H(P||seed_{i,k})$;
  // derive a key using the chosen bytes
  $r_{i,k} \xleftarrow{\$} \{0,1\}^\ell$;
  $c_{i,k}^{(1)} = \mathsf{E}(r_{i,k}, \tau_i), c_{i,k}^{(2)} = r_{i,k} \oplus key_{i,k}$;
  // encrypt m_i's tag under the key
  $c_{i,k} = \{c_{i,k}^{(1)}, c_{i,k}^{(2)}\}, p_{i,k} = \{c_{i,k}, ind_{i,k}^{(1)}, ..., ind_{i,k}^{(\alpha)}\}$;
end
return $p_i = \{p_{i,1}, ..., p_{i,l}\}$;

- 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.

- 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.

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 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, w denote the number of unfilled positions of $\overrightarrow{BF}$ in τ's item set, then*

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

where ι can be computed by $\mathcal{CS}$: it maintains a counter for the total number of insertions and subtracts t from the total number.

The proof of Theorem 1 is deferred to Appendix B.1.

7 Further discussions

Limitations. 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 this 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 employing the secure comparison technique to compare a private frequency with a threshold without leaking any additional information [35, 36, 55], 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.

Moreover, EpiOracle achieves probabilistic frequency statistics. A possible improvement could be to increase the accuracy of the FuzDet and Increase algorithms within it. However, we stress that even deterministic algorithms cannot achieve definite early warning for epidemics. 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. This is a non-technical limitation that any early warning system, including our scheme and practical systems [13, 14], encounters.

Scalability. Scalability of EpiOralce can be discussed from the aspects of dynamic facilities and geographic area of deployment.

Dynamic facilities. In real-world scenarios, new healthcare facilities could join the early epidemic warning system, while existing ones may leave. EpiOracle supports flexible joining and leaving when such changes occur. Specifically, for both staying and newly joined facilities, no additional operations are required, as 1) the participating facilities do not directly interact with each other and 2) no shared secret exists between a facility and the cloud; for the cloud, the only required update is to change the permissions for joining and leaving facilities, which should be handled by the underlying authentication mechanism.

Geographic area of deployment. 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.

Deployment. Similar to practical early warning systems like NSSP, EpiOracle can be deployed and managed by an authority (e.g., CDC). On the facility side, the EpiOralce application is operated by the department affiliated with the authority. At designated intervals (e.g., daily or weekly), symptom lists are collected and inputted into EpiOracle by the department.

Similarity metric. As discussed in Section 3, we choose Hamming distance as the similarity metric for the encoded symptom lists. 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 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 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 healthcare facilities.

In addition to Hamming distance, alternative similarity metrics, such as set difference and edit distance, can also be used to instantiate EpiOracle. We briefly discuss how to achieve fuzzy detection based on set difference as an example here, and details of this approach can be found in [37].

Given a symptom list $m_1 = \{m_{1,1}, \ldots, m_{1,n}\}$, where $m_{1,i} (i = 1, \ldots, n)$ is a symptom, randomly choose τ_1 as its tag and represent it as a polynomial f of degree at most $n - 2t - 1$ using Reed-Solomon code; generate another polynomial $f'(x) = \sum_{i=0}^{n-1} a_i x^i + x^n$ such that $f'(m_{1,i}) = f(m_{1,i})$ for all $i \in [1, n]$; publish $\{a_i\}_{i=1}^{n-1}$ as a helper parameter. Subsequently, when another symptom list $m_2 = \{m_{2,1}, \ldots, m_{2,n'}\}$ is generated, f (i.e., τ_1) can be recovered using the helper parameter iff m_2 agrees with m_1 in at least $n - t$ positions. By doing so, m_1 and m_2 can be mapped to the same tag if their set difference is $\leq t$, which achieves fuzzy detection.

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 the 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 $|m_i|$ and compatible with lists of all lengths.

More discussions are deferred to Appendix A.

8 Security analysis

We analyze the security of EpiOracle in terms of the privacy of symptom and frequency. The proofs of the following theorems are sketched below and detailed in Appendix B.2 and Appendix B.3.

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

Proof sketch. We prove the above theorem by demonstrating that given the view of the cloud server in the interaction with a healthcare facility, for any probabilistic polynomial-time (PPT) adversary, he can distinguish the view between some random strings with only negligible probability.

In the proof, we introduce a sequence of games. The first game is identical with IND\$-CDA$^{b=1}_{\mathcal{A},\mathcal{M}}$, i.e., a PPT adversary $\mathcal{A}$ is given all information that can be extracted by the cloud server about the symptom content. The last game is identical with IND\$-CDA$^{b=0}_{\mathcal{A},\mathcal{M}}$, i.e., $\mathcal{A}$ is given some random strings that reveal nothing about the symptom content. With the fundamental lemma of game-playing (Lemma 1 in [25]), we demonstrate that the difference between the advantages of $\mathcal{A}$ in the first game and the last game is negligible.

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

Proof sketch. We prove this theorem by demonstrating that, fixing two symptom lists and updating their counter(s), any PPT adversary can tell whether these two updates are conducted on the same counter only with negligible probability.

In IND-RTA, when $d = 0$, i.e., the Hamming distance between these two symptom lists is 0 for $b = 1$, the advantage of the adversary $\mathcal{A}$ is maximized. In this case, the advantage of $\mathcal{A}$ in IND-RTA is no larger than twice the advantage of another PPT adversary $\mathcal{A}'$ in IND\$-CDA, i.e., $\mathrm{Adv}^{\text{IND-RTA}}_{\mathcal{A},\mathcal{M}}(\ell) \leq 2 \cdot \mathrm{Adv}^{\text{IND\$-CDA}}_{\mathcal{A}',\mathcal{M}}(\ell)$.

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 detail a server-aided scheme [51] to resist such an adversary in Appendix C.

9 Implementation and evaluation

9.1 Implementation

We implement an EpiOracle prototype in Java with JPBC library [2], and the source code is available at

https://github.com/Yuan-Zhang-uestc/EpiOracleCode.

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 $\overrightarrow{BF}$. 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 $SimRatio \cdot |m_i|$. Not only the symptom lists themselves but their all possible permutations are compared in fuzzy detection.

1

9.2 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 conduct experiments on three datasets: 1) a real-world dataset consisting

of 200 symptom lists associated with COVID-19, 2) a larger sample of synthesized lists associated with COVID-19 (details of the synthesis process will be provided later), and 3) the combination of a set of synthesized lists associated with COVID-19 and "noisy" lists comprising various symptoms potentially indicative of other diseases. In these experiments, each list is processed using **FuzDet** and **HelParGen** to generate a tag, and the maximum number of lists sharing the same tag is calculated and compared with the ideal value, i.e. the total number of lists associated with COVID-19.

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.

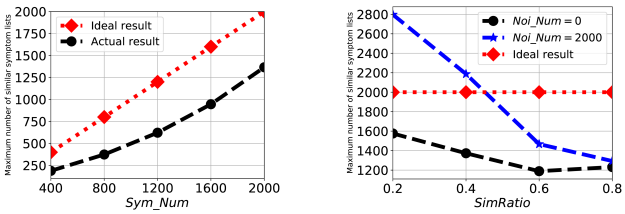
Now we present how we synthesize the symptom lists used for evaluation. According to the statistical proportions of the symptoms reported by WHO [7], we generate 2000 symptom lists associated with COVID-19 and encode them using SNOMED CT. We initialize 2000 empty lists and, for each symptom and its proportion 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 16 GB LPDDR4X RAM. The variants in the following experiments are summarized in Table 2.

9.3 Accuracy

Fuzzy detection. We first conduct experiments on the 200 real-world symptom lists. With this dataset as input, the number of inputted symptom lists remains constant, and theoretically, the experimental results fluctuate depending on the values of l and $SimRatio$. To evaluate the accuracy fuzzy detection under varying values of these two parameters, we conduct experiments with different $(l, SimRatio)$ and present the results in Table 3. According to the results, the accuracy of fuzzy detection of EpiOracle is around 70% when $(l, SimRatio) = (10, 0.8)$.

Table 2: Notation summary.

Variant	Explanation
l	number of rounds in helper parameter generation
α	number of positions sampled from a symptom list
$SimRatio$	α/(length of the symptom list)
Sym_Num	number of symptom lists associated with COVID-19
Noi_Num	number of noisy symptom lists
s	number of slots in an item set
L	number of slots of the Bloom Filter
t	threshold/number of insertions performed on a target tag
ι	number of the insertions performed on other tags
T	explicit threshold



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

Figure 7: Results of fuzzy detection.

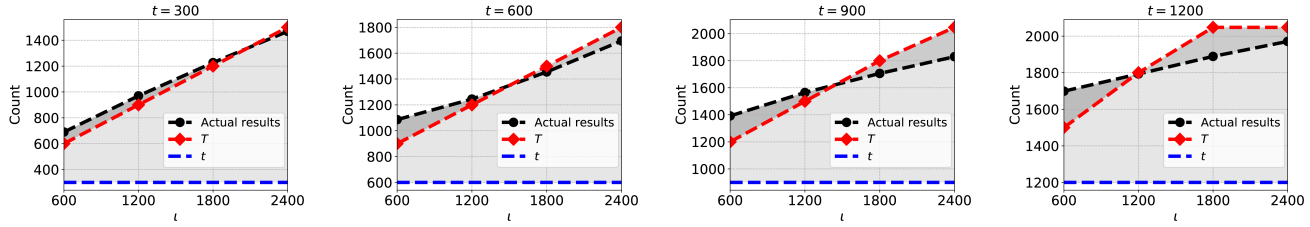
To evaluate the accuracy of fuzzy detection on a larger dataset, we conducted experiments using synthesized lists. To show the probability of false negative in fuzzy detection, we use only lists associated with COVID-19 as input and depict the experimental results in Figure 7a and Figure 7b, varying values of Sym_Num and $SimRatio$, respectively. Furthermore, to show the probability of false positive in fuzzy detection, we take both lists associated with COVID-19 and noisy lists as input, compare the experimental results with the ones obtained when using only COVID-19 lists, and present the comparison in Figure 7b while varying $SimRatio$.

The results indicate that the accuracy of fuzzy detection diminishes as $SimRatio$ increases in the absence of noise. However, in practical scenarios, where symptom lists of other diseases are also present, 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.

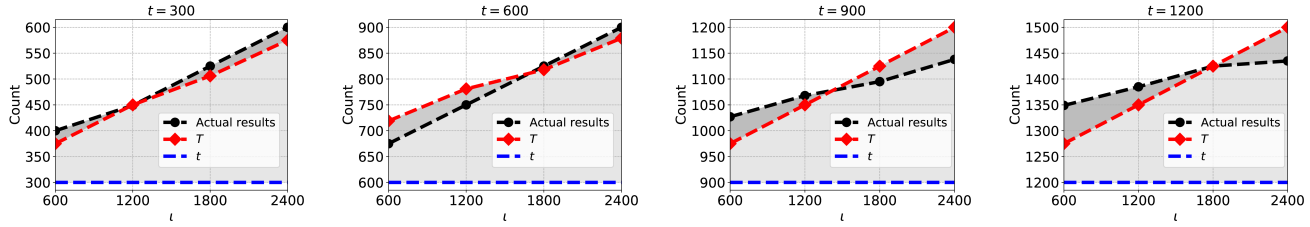
Increment count. The count of a tag is increased by inserting the tag into the Bloom Filter, i.e., randomly choosing an unfilled slot of the tag's item set and setting the slot to 1. Later, the count can be retrieved via the number of filled slots. To evaluate the accuracy of the increment count, we simulate the case where a target tag has been inserted for t times and other tags have been inserted for ι times; count the occurrences of the target tag; and compare the results with ideal ones (i.e. T).

Table 3: Experimental results of fuzzy detection based on 200 real-world symptom lists.

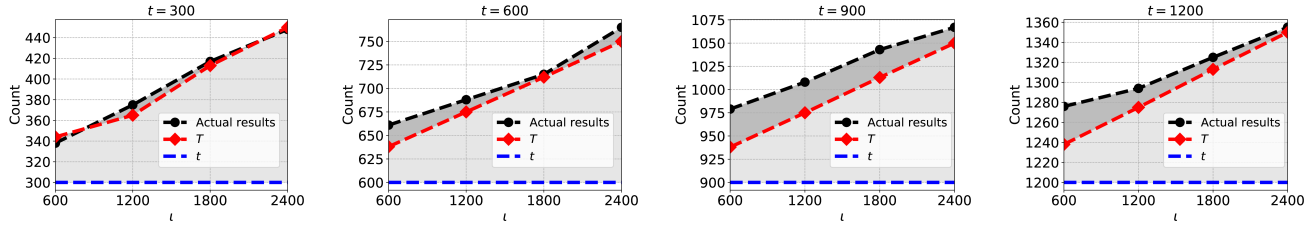
$SimRatio$	l	Number of similar lists detected by EpiOracle				
0.6	1	136	109	92	57	75
	5	123	130	86	89	125
	10	97	131	85	123	131
0.7	1	77	93	83	85	142
	5	144	161	142	136	69
	10	146	142	141	141	141
0.8	1	75	64	107	64	78
	5	141	142	138	142	145
	10	140	144	141	141	141



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

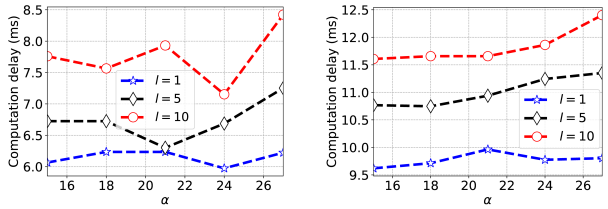


(b) $L = 2^{14}, s = 2^{11}$.



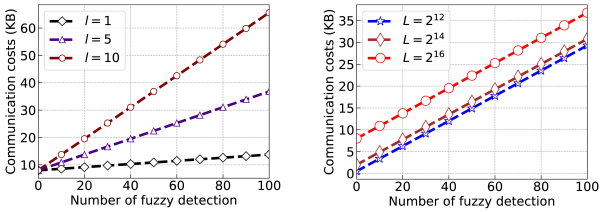
(c) $L = 2^{16}, s = 2^{12}$.

Figure 8: Evaluation results of increment count.



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

Figure 9: Computation delay on the facility side.



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

Figure 10: Communication costs between a facility and the cloud server.

We present the value of T in Table 4 and the experimental results in Figure 8. In the experiments, we vary the values of (s, L, t, ι) to present the accuracy of the increment count in various parameter settings. The results demonstrate that using T as the explicit threshold instead of t can significantly improve the accuracy. 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.

9.4 Efficiency

Computation costs. On the facility side, the computation costs increase with the values of l, α, L, and s. Therefore, we conduct 2000 experiments with different (l, α, L, s) configurations and present the average computation delay in Figure 9. Notably, when $(l, \alpha, L, s) = (10, 27, 2^{16}, 2^{12})$, the average computation delay is around 12.4 ms.

To further demonstrate the practicality of EpiOracle in terms of computation costs, we also evaluate the computation delay of the plain process: the frequency of symptom lists is determined by computing the Hamming distance between each pair of plaintext

Table 4: Explicit threshold (T).

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

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

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

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

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

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

lists. Given 2000 symptom lists, we computed the minimum Hamming distance across all permutations for each pair. Based on these distances, we then obtain the frequencies of these 2000 lists. The average time delay for obtaining one frequency is around 27 ms. Therefore, the computation costs of EpiOracle on the facility side are considered acceptable.

Communication costs. The communication costs between a healthcare facility and the cloud server arise from transmitting the Bloom Filter, helper parameters, and hash values of tags. Therefore, to evaluate the communication costs in EpiOralce, 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 Figure 10. 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.

10 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.

Acknowledgments

This work was supported in part by the National Key Research and Development Program of China under Grant 2023YFB3106503; in part by the National Nature Science Foundation of China under Grant 62472074; in part by the Young Elite Scientists Sponsorship Program by China Association for Science and Technology (CAST) under Grant 2022QNR-C001; and in part by Sichuan Science and Technology Program under Grant 2022ZDZX0038, Grant 2023ZYD0142, and Grant 24NSFSC2271.

References

[1] 2008. 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] 2018. The Java Pairing Based Cryptography Library (JPBC). http://gas.dia.unisa.it/projects/jpbc/#.Yg9TnC-KFqs
[3] 2019. 2009 H1N1 Pandemic. https://www.cdc.gov/flu/pandemic-resources/2009-h1n1-pandemic.html
[4] 2019. 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
[5] 2020. Coronavirus: The psychology of panic buying. https://www.bbc.com/worklife/article/20200304-coronavirus-covid-19-update-why-people-are-stockpiling
[6] 2020. Guide for the BioSense platform. https://www.cdc.gov/nssp/biosense/onboarding-guide/pdf/New-Site-OG-508.pdf
[7] 2020. *Report of the WHO-China joint mission on coronavirus disease 2019 (COVID-19).* Technical Report. World Health Organization (WHO). https://www.who.int/publications/i/item/report-of-the-who-china-joint-mission-on-coronavirus-disease-2019-(covid-19)
[8] 2020. '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
[9] 2022. China Covid: 'Panic-buying' and shortages as restrictions are eased. https://www.bbc.com/news/world-asia-china-63976197
[10] 2022. ICD-11. https://icd.who.int
[11] 2022. 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
[12] 2023. National Outbreak Report System. https://www.cdc.gov/nors/index.html
[13] 2023. National Syndromic Surveillance Program (NSSP). https://www.cdc.gov/nssp/how-sys.html
[14] 2023. ProMED-mail. https://promedmail.org
[15] 2023. SNOMED CT. https://www.snomed.org
[16] 2023. WHO Coronavirus (COVID-19) dashboard. https://covid19.who.int
[17] Shashank Agrawal, Peihan Miao, Payman Mohassel, and Pratyay Mukherjee. 2018. PASTA: password-based threshold authentication. In *Proc. CCS*. 2042–2059.
[18] Adil Ahmad, Juhee Kim, Jaebaek Seo, Insik Shin, Pedro Fonseca, and Byoungyoung Lee. 2021. CHANCEL: Efficient Multi-client Isolation Under Adversarial Programs.. In *Proc. NDSS*.
[19] FIDO Alliance. 2012. What is FIDO? https://fidoalliance.org/what-is-fido/
[20] Emily Alsentzer, Sarah-Blythe Ballard, Joan Neyra, Delphis M Vera, Victor B Osorio, Jose Quispe, David L Blazes, and Luis Loayza. 2020. Assessing 3 outbreak detection algorithms in an electronic syndromic surveillance system in a resource-limited setting. *Emerging Infectious Diseases* 26, 9 (2020), 2196.
[21] Gilad Asharov, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Ariel Nof, Benny Pinkas, Katsumi Takahashi, and Junichi Tomida. 2022. Efficient Secure Three-Party Sorting with Applications to Data Analysis and Heavy Hitters. In *Proc. CCS*. 125–138.
[22] Borja Balle and Yu-Xiang Wang. 2018. Improving the gaussian mechanism for differential privacy: Analytical calibration and optimal denoising. In *Proc. ICML*. 394–403.
[23] Carsten Baum, Tore Frederiksen, Julia Hesse, Anja Lehmann, and Avishay Yanai. 2020. PESTO: proactively secure distributed single sign-on, or how to trust a hacked server. In *Proc. EuroS&P*. 587–606.
[24] Mihir Bellare, Sriram Keelveedhi, and Thomas Ristenpart. 2013. Message-locked encryption and secure deduplication. In *Proc. EUROCRYPT*. 296–312.
[25] Mihir Bellare and Phillip Rogaway. 2006. The security of triple encryption and a framework for code-based game-playing proofs. In *Proc. EUROCRYPT*. 409–426.
[26] Burton H Bloom. 1970. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM* 13, 7 (1970), 422–426.
[27] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. 2021. Lightweight techniques for private heavy hitters. In *Proc. S&P*. 762–776.

[28] Dan Boneh, Dmitry Kogan, and Katharine Woo. 2020. Oblivious pseudorandom functions from isogenies. In *Proc. ASIACRYPT*. 520–550.
[29] Dan Boneh, Hart William Montgomery, and Ananth Raghunathan. 2010. Algebraic pseudorandom functions with improved efficiency from the augmented cascade. In *Proc. CCS*. 131–140.
[30] Ran Canetti, Benjamin Fuller, Omer Paneth, Leonid Reyzin, and Adam Smith. 2021. Reusable fuzzy extractors for low-entropy distributions. *Journal of Cryptology* 34, 1 (2021), 1–33.
[31] Ran Canetti, Yael Tauman Kalai, Mayank Varia, and Daniel Wichs. 2010. On symmetric encryption and point obfuscation. In *Proc. TCC*. 52–71.
[32] Sylvain Chatel, Apostolos Pyrgelis, Juan Ramón Troncoso-Pastoriza, and Jean-Pierre Hubaux. 2021. Privacy and integrity preserving computations with CRISP. In *Proc. USENIX Security*. 2111–2128.
[33] Long Chen, Ya-Nan Li, Qiang Tang, and Moti Yung. 2022. End-to-same-end encryption: modularly augmenting an app with an efficient, portable, and blind cloud storage. In *Proc. USENIX Security*. 2353–2370.
[34] Nanshan Chen, Min Zhou, Xuan Dong, Jieming Qu, Fengyun Gong, Yang Han, Yang Qiu, Jingli Wang, Ying Liu, Yuan Wei, et al. 2020. Epidemiological and clinical characteristics of 99 cases of 2019 novel coronavirus pneumonia in Wuhan, China: a descriptive study. *The Lancet* 395, 10223 (2020), 507–513.
[35] Anders Dalskov, Daniel Escudero, and Marcel Keller. 2021. Fantastic four:{Honest-Majority}{Four-Party} secure computation with malicious security. In *Proc. USENIX Security*. 2183–2200.
[36] Ivan Damgard, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. 2006. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *Proc. TCC*. 285–304.
[37] Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. 2004. Fuzzy extractors: how to generate strong keys from biometrics and other noisy data. In *Proc. EUROCRYPT*. 523–540.
[38] eCQI Resource Center. 2023. Fast Healthcare Interoperability Resources (FHIR). https://www.hl7.org/fhir/security.html#authentication
[39] Centers for Disease Control and Prevention. 2012. Unexplained Respiratory Disease Outbreak working group activities-worldwide, March 2007-September 2011. *MMWR. Morbidity and Mortality Weekly Report* 61, 26 (2012), 480–483.
[40] International Organization for Standardization. 2022. ISO/IEC 27002:2022 Information security, cybersecurity and privacy protection — Information security controls. https://www.iso.org/standard/75652.html
[41] Michael J Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. 2005. Keyword search and oblivious pseudorandom functions. In *Proc. PKC*. 303–324.
[42] Deborah W Gould, David Walker, and Paula W Yoon. 2017. The evolution of BioSense: lessons learned and future directions. *Public Health Reports* 132, 1_suppl (2017), 7S–11S.
[43] Marcus Hähnel, Weidong Cui, and Marcus Peinado. 2017. High-Resolution Side Channels for Untrusted Operating Systems. In *Proc. USENIX ATC*. 299–312.
[44] Dick Hardt. 2012. The OAuth 2.0 Authorization Framework. https://www.rfc-editor.org/rfc/rfc6749.html
[45] Carmit Hazay, Emmanuela Orsini, Peter Scholl, and Eduardo Soria-Vazquez. 2022. Tinykeys: a new approach to efficient multi-party computation. *Journal of Cryptology* 35, 2 (2022), 13.
[46] Yan Huang, David Evans, Jonathan Katz, and Lior Malka. 2011. Faster secure {Two-Party} computation using garbled circuits. In *Proc. USENIX Security*.
[47] Thomas Humphries, Rasoul Akhavan Mahdavi, Shannon Veitch, and Florian Kerschbaum. 2022. Selective MPC: Distributed Computation of Differentially Private Key-Value Statistics. In *Proc. CCS*. 1459–1472.
[48] Tyler Hunt, Zhiting Zhu, Yuanzhong Xu, Simon Peter, and Emmett Witchel. 2016. Ryoan: A distributed Sandbox for untrusted computation on secret data. In *Proc. OSDI*. 533–549.
[49] Lori Hutwagner, William Thompson, G Matthew Seeman, and Tracee Treadwell. 2003. The bioterrorism preparedness and response early aberration reporting system (EARS). *Journal of Urban Health* 80 (2003), i89–i96.
[50] Ayman Jarrous and Benny Pinkas. 2009. Secure hamming distance based computation and its applications. In *Proc. ACNS*. 107–124.
[51] Sriram Keelveedhi, Mihir Bellare, and Thomas Ristenpart. 2013. DupLESS:server-aided encryption for deduplicated storage. In *Proc. USENIX Security)*. 179–194.
[52] Mathias Lecuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. 2019. Certified robustness to adversarial examples with differential privacy. In *Proc. S&P*. 656–672.
[53] JaeHyuk Lee, Jinsoo Jang, Yeongjin Jang, Nohyun Kwak, Yeseul Choi, Changho Choi, Taesoo Kim, and Brent Byunghoon Kang. 2017. Hacking in darkness: Return-oriented programming against secure enclaves. (2017), 523–539.
[54] Linsheng Liu, Daniel S Roche, Austin Theriault, and Arkady Yerukhimovich. 2022. Fighting fake news in encrypted messaging with the fuzzy anonymous complaint tally system (FACTS). In *Proc. NDSS*.
[55] Eleftheria Makri, Dragos Rotaru, Frederik Vercauteren, and Sameer Wagh. 2021. Rabbit: Efficient comparison for secure multi-party computation. In *Proc. FC*. 249–270.
[56] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R Savagaonkar. 2013. Innovative instructions and software model for isolated execution. *Hasp@ isca* 10, 1 (2013).
[57] 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. 2022. Monkeypox outbreak—nine states, May 2022. *Morbidity and Mortality Weekly Report* 71, 23 (2022), 764.
[58] NIST. 2020. Security and Privacy Controls for Information Systems and Organizations. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r5.pdf
[59] Mohammad Norouzi, David J Fleet, and Russ R Salakhutdinov. 2012. Hamming distance metric learning. *Advances in Neural Information Processing Systems* 25 (2012).
[60] Margarita Osadchy, Benny Pinkas, Ayman Jarrous, and Boaz Moskovich. 2010. Scifi-a system for secure face identification. In *Proc. S&P*. 239–254.
[61] Sarvar Patel, Giuseppe Persiano, Kevin Yeo, and Moti Yung. 2019. Mitigating leakage in secure cloud-hosted data structures: Volume-hiding for multi-maps via hashing. In *Proc. CCS*. 79–93.
[62] Edo Roth, Daniel Noble, Brett Hemenway Falk, and Andreas Haeberlen. 2019. Honeycrisp: large-scale differentially private aggregation without a trusted core. In *Proc. SOSP*. 196–210.
[63] Edo Roth, Hengchu Zhang, Andreas Haeberlen, and Benjamin C Pierce. 2020. Orchard: Differentially private analytics at scale. In *Proc. OSDI*. 1065–1081.
[64] Felix Schuster, Manuel Costa, Cédric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, and Mark Russinovich. 2015. VC3: Trustworthy data analytics in the cloud using SGX. In *Proc. S&P*. 38–54.
[65] Dan Simon, Bernard Aboba, and Ryan Hurst. 2008. The EAP-TLS Authentication Protocol. https://datatracker.ietf.org/doc/html/rfc5216#section-2.1.2
[66] Alex Skvortsov and Branko Ristic. 2012. Monitoring and prediction of an epidemic outbreak using syndromic observations. *Mathematical Biosciences* 240, 1 (2012), 12–19.
[67] Flavio Toffalini, Mariano Graziano, Mauro Conti, and Jianying Zhou. 2021. SnakeGX: A sneaky attack against SGX Enclaves. In *Proc. ACNS*. 333–362.
[68] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. 2018. Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution. In *Proc. USENIX Security*. 991–1008.
[69] Longde Wang, Yu Wang, Shuigao Jin, Zunyou Wu, Daniel P Chin, Jeffrey P Koplan, and Mary Elizabeth Wilson. 2008. Emergence and control of infectious diseases in China. *The Lancet* 372, 9649 (2008), 1598–1605.
[70] Weier Wang, Jianming Tang, and Fangqiang Wei. 2020. Updated understanding of the outbreak of 2019 novel coronavirus (2019-nCoV) in Wuhan, China. *Journal of Medical Virology* 92, 4 (2020), 441–447.
[71] Zunyou Wu and Jennifer M McGoogan. 2020. 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 (2020), 1239–1242.
[72] Qingqing Ye, Haibo Hu, Xiaofeng Meng, and Huadi Zheng. 2019. PrivKV: key-value data collection with local differential privacy. In *Proc. S&P*. 317–331.
[73] Mingxun Zhou, Tianhao Wang, TH Hubert Chan, Giulia Fanti, and Elaine Shi. 2022. Locally differentially private sparse vector aggregation. In *Proc. S&P*. 422–439.

A Practical considerations

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 ones 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.

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 [49].

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 [71]). 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.

B Proofs

B.1 Proof of Theorem 1

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

$$p_w = \frac{\begin{pmatrix} s \\ s-w \end{pmatrix} \cdot \begin{pmatrix} L-s \\ \iota-s+w \end{pmatrix}}{\begin{pmatrix} L \\ \iota \end{pmatrix}}.$$

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. □

B.2 Proof of Theorem 2

Proof. To prove **Theorem 2**, we first introduce 6 games presented in Figure 11 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 is identical with IND\$-CDA$^{b=1}_{\mathcal{A},\mathcal{M}}$. 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

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

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

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

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

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

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

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

where $\text{Adv}^{\text{ROR}}_{\mathcal{A}',\Pi}(\ell)$ is the advantage of a PPT adversary $\mathcal{A}'$ in the real-or-random (ROR) game for π defined in Figure 4 of Section 5.3.

$\mathcal{A}'$ 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 $\tau \xleftarrow{\$} \{0,1\}^\ell$.

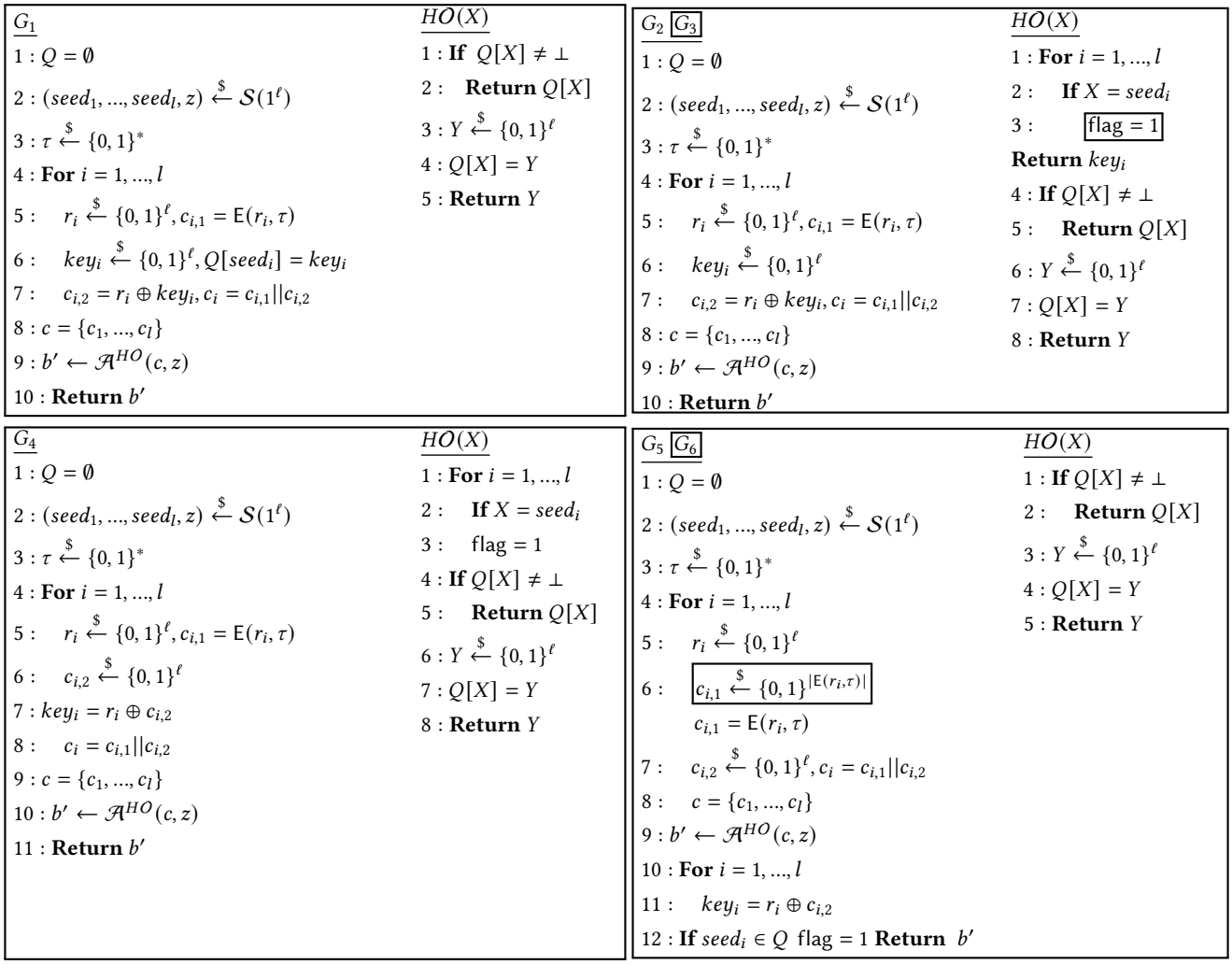


Figure 11: Game used in the proof of Theorem 2.

(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 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,

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

Furthermore, we have

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

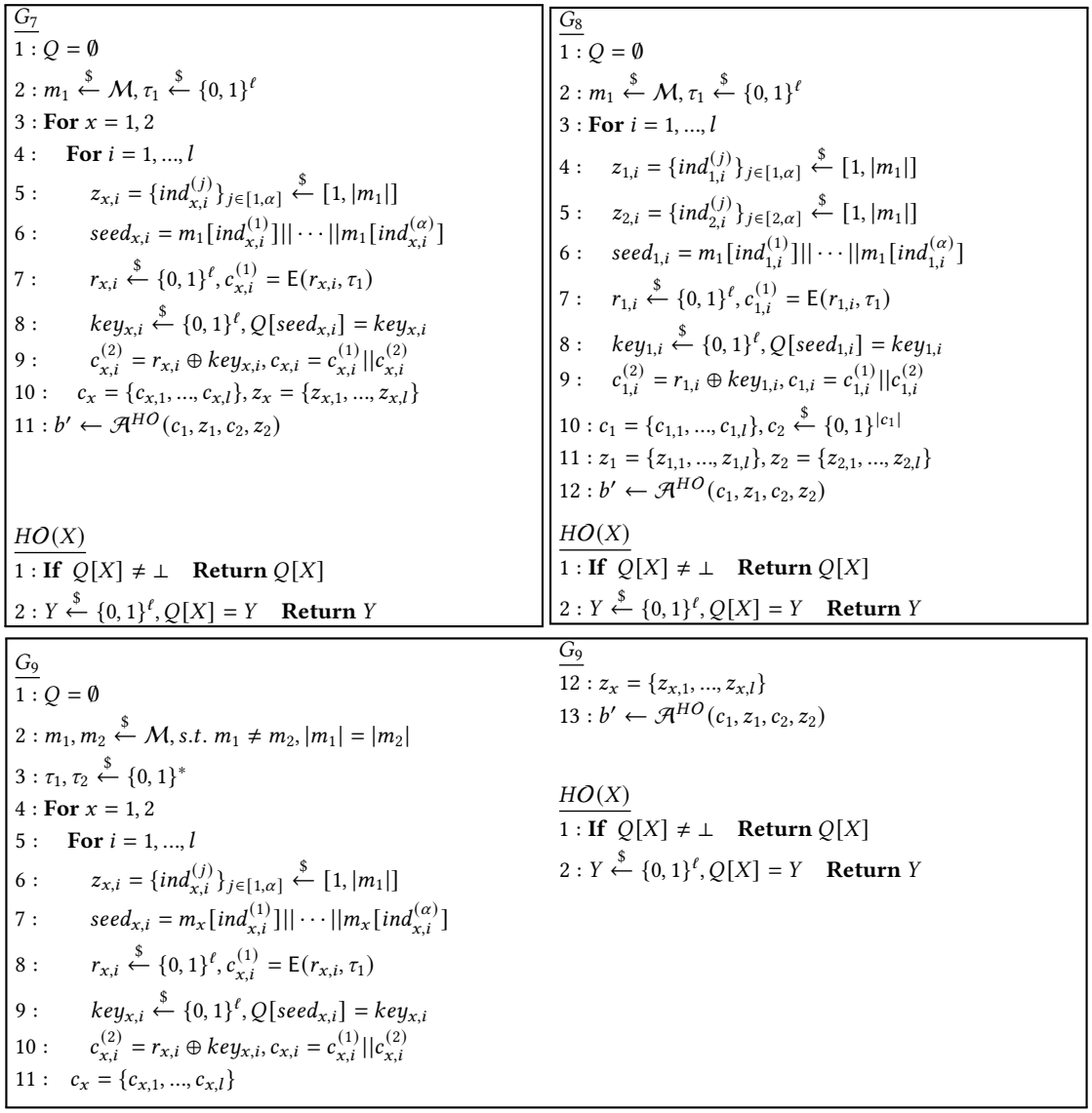


Figure 12: Game used in the proof of Theorem 3.

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

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

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

This concludes the proof. □

B.3 Proof of Theorem 3

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, $\Pr[\text{IND-RTA}_{\mathcal{A},\mathcal{M}} \Rightarrow 1] \leq \Pr[\text{IND-RTA}_{\mathcal{A},\mathcal{M}} \Rightarrow 1|d = 0]$. Moreover, G_7 shown in Figure 12 is identical with $\text{IND-RTA}_{\mathcal{A},\mathcal{M}}^{d=0,b=1}$, then

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

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

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

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

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

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

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

This concludes the proof. □

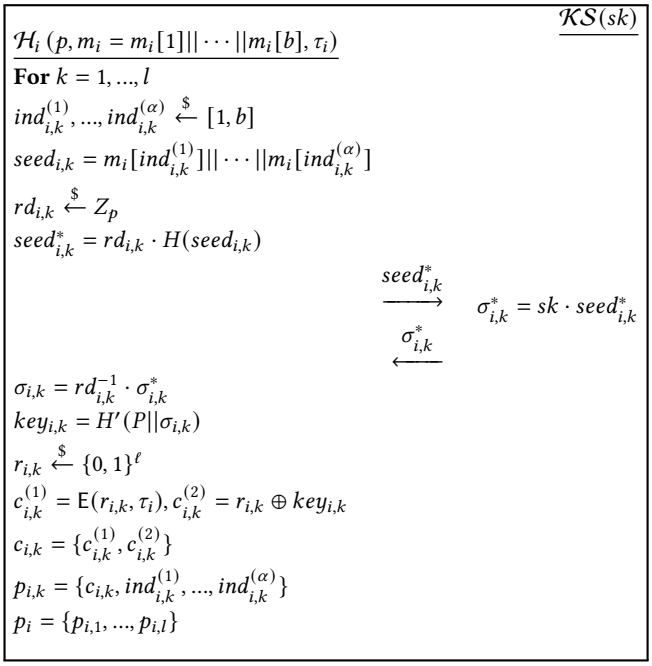


Figure 13: Server-aided helper parameter generation.

C Security enhancement

As shown in Figure 13, 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 $ind_{i,k}^{(1)}, ..., ind_{i,k}^{(\alpha)}$, $k \in [1, l]$ contained in p_i, $\mathcal{H}_j$ computes $seed'_{i,k}$ as $m_j[ind_{i,k}^{(1)}]||\cdots||m_j[ind_{i,k}^{(\alpha)}]$, 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.

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 [17, 23] 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, 41].

D Artifact

Abstract

Our artifact consists of an EpiOracle prototype and a synthesized 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 9.

Content

The artifact comprises the following sub-directories:

- ./EpiOracle, which contains the sourcecode of the EpiOracle prototype.
- ./DataSet, which contains the synthesized data set used in the experiments detailed in Section 9.
- 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://github.com/Yuan-Zhang-uestc/EpiOracleCode/blob/main/README.md).