Research Article

PvCT: A Publicly Verifiable Contact Tracing Algorithm in Cloud Computing

Yixiao Zhu ¹, Wenjie Ma ¹, Jiangtao Cui ^{1,2}, Xiaofang Xia ¹, Yanguo Peng ^{1,2},
and Jianting Ning ^{3,4}

¹School of Computer Science and Technology, Xidian University, Xi'an 710071, China

²Qingdao Institute of Computing Technology, Xidian University, Qingdao 266109, China

³College of Mathematics and Informatics, Fujian Normal University, Fuzhou 350007, China

⁴The State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China

Correspondence should be addressed to Jiangtao Cui; cuijt@xidian.edu.cn

Received 12 January 2021; Accepted 3 May 2021; Published 2 June 2021

Academic Editor: M.A. Jabbar

Copyright © 2021 Yixiao Zhu et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Contact tracing is a critical tool in containing epidemics such as COVID-19. Researchers have carried out a lot of work on contact tracing. However, almost all of the existing works assume that their clients and authorities have large storage space and powerful computation capability and clients can implement contact tracing on their own mobile devices such as mobile phones, tablet computers, and wearable computers. With the widespread outbreaks of the epidemics, these approaches are of less robustness to a larger scale of datasets when it comes to resource-constrained clients. To address this limitation, we propose a publicly verifiable contact tracing algorithm in cloud computing (PvCT), which utilizes cloud services to provide storage and computation capability in contact tracing. To guarantee the integrity and accuracy of contact tracing results, PvCT applies a novel set accumulator-based authentication data structure whose computation is outsourced, and the client can check whether returned results are valid. Furthermore, we provide rigorous security proof of our algorithm based on the q -Strong Bilinear Diffie–Hellman assumption. Detailed experimental evaluation is also conducted on three real-world datasets. The results show that our algorithm is feasible within milliseconds of client CPU time and can significantly reduce the storage overhead from the size of datasets to a constant 128 bytes.

1. Introduction

In public health domain, contact tracing is a critical approach for identifying people who may have come into contact with diagnosed people infected with some epidemics such as Ebola virus disease, H1N1 influenza pandemic, or coronavirus disease 2019 (COVID-19). By tracing the contacts of infected individuals and treating them appropriately based on their testing results, public health departments can contain and mitigate the community transmission of infectious diseases. In history, contact tracing is commonly used as an important tool to fight against epidemics. For example, during the 2014–2016 outbreak of Ebola in West Africa, the World Health Organization issued guidelines to conduct contact tracing for breaking transmission chains of the Ebola virus [1].

Nowadays, contact tracing is also playing a critical role in all our efforts to contain the ongoing COVID-19 pandemic.

Researchers have carried out a lot of works on contact tracing [2–8]. For instance, in [4], the authors put forward a privacy-preserving system for contact tracing based on secure two-party private set intersection cardinality technique. In [6], the authors develop a Bluetooth-based contact tracing system. In [7], the authors mainly focus on privacy leakage, and they propose a blockchain-based privacy-preserving contact tracing algorithm. The authors in [8] propose a contact tracing algorithm with access control, which guarantees that only authorized people can execute contact tracing process. However, almost all of the existing works assume that their clients and authorities have large storage space and powerful computation capability and clients can implement contact tracing on their own mobile devices such

as mobile phones, tablet computers, and wearable computers. In these literatures, the authorities such as hospitals or Centers for Disease Control and Prevention (CDC) are required to store all the travel records of diagnosed people, and the clients are demanded to store all their travel records at their own sides.

However, with the spread of diseases, the number of travel records of diagnosed people grows rapidly. Meanwhile, besides classical travel records such as accurate location and relevant timestamp, there is a special category of travel records, the transportation data. It contains information such as train/flight number, license plates, and other important information which is also quite useful in the contact tracing process. Meanwhile, the more complete these travel records are, the more comprehensive the contact tracing can be. In applications, all the travel records and transportation data are collected through mobile crowd-sensing (MCS) technology, which is capable of sensing and collecting data using various mobile devices belonging to all the clients and authorities.

These terminal devices have certain limitations. First, they have limited hardware resources in storage and computation. These restrictions make those mobile devices unbearable to the large storage and computation burden incurred by the rapid increment of travel records scale. Second, it is also difficult to synchronize all the data as well as collaborate all the computation among the clients if there are only terminal devices. If clients are organizations such as colleges, synchronization delay can adversely affect the accuracy and efficiency of disease control over all of their students and staff. That is, the latency can directly lead to the failure of epidemic prevention. Hence, for clients and authorities, they need other ways to deal with the tricky situation introduced by limited mobile devices. To the best of our knowledge, there are no prior works that can be applied to such a scenario where clients are resource-constrained and efficient management is required. In this paper, considering that, with cloud computing service provided by certain cloud service providers (CSPs) such as AWS and Azure, the client can pay for the resource he/she lacks and release it once his/her work is finished, and we resort to CSPs for storage and computation assistance.

Nevertheless, it brings up new issues when introducing CSPs:

- (1) CSPs can be compromised by a malicious adversary [9]
- (2) Even if CSPs honestly follow the clients' rules, there still exist various problems such as program malfunction, data loss, or some unintended mistakes

Therefore, CSPs cannot be fully trusted, and there is no guarantee that the returned results are correct and complete. However, in the contact tracing scenario, the accuracy and integrity of the returned result are of vital importance and sometimes matter tens of thousands of lives. Assume that someone is a close contact of one COVID-19 diagnosed person, but CSPs return a false negative contact tracing result which indicates that he is not. In this case, not only he/

she cannot obtain a timely treatment but also the people who are close contacts of him/her cannot be tracked anymore. This hinders the process to contain the pandemic.

In this paper, we aim to solve the problem that how to achieve both accuracy and integrity of contact tracing results in untrusted cloud computing setting. Currently, the mainstream method (if not the only method) to satisfy the above requirements is to take advantage of various verifiable computation techniques. A possible solution is to utilize general verifiable computation schemes [10, 11] by using techniques such as succinct noninteractive argument of knowledge (SNARK). However, almost all existing SNARK algorithms are too complicated to be deployed in practice [12]. Thus, in this paper, we adopt another kind of technique which enables ad-hoc verifiable computation through constructing an authentication data structure (ADS) [13, 14]. Briefly speaking, an ADS is a data structure whose computation can be outsourced to an untrusted server and the client can check if the returned result is valid.

There are some ADS-based query authentication techniques studied for outsourced databases [15–17]. However, there still exist several main challenges, making the conventional schemes inapplicable to contact tracing. First, the conventional schemes rely heavily on one data owner signing the ADS using a secret key. In contrast, in the contact tracing scenario, there are two data owners and two phases, one is the query phase and the other is the matching phase. Especially in the matching phase, only the authority can append new records of diagnosed people to its database, and a client cannot act as the authority in this phase because he does not have the authority's secret key and cannot sign its ADS. Second, a traditional ADS is constructed on a fixed dataset, and such an ADS cannot be efficiently adapted to a contact tracing scenario in which the data are unbounded with the spread of the disease. Third, in conventional outsourced databases, ADS is always regenerated to support more queries. However, it is difficult for clients who have limited resources to implement. Thus, a more generic ADS is preferable to support different phases that may happen in contact tracing scenario.

To address this issue, we propose a novel set accumulator-based ADS scheme that enables public verification over contact tracing, guaranteeing both accuracy and integrity check. And, hence, on that basis, we propose a novel framework called publicly verifiable contact tracing (PvCT), which employs publicly verifiable computation techniques to guarantee both integrity and accuracy of contact tracing result. More specifically, we provide each client and authority with an additional ADS. And, based on this ADS, untrusted CSP can construct and return a cryptographic proof, known as *verification object* (VO), for clients to verify the result of contact tracing. The information flows among CSPs, clients, and authorities, as illustrated in Figure 1.

To summarize, our contributions made in this paper are as follows:

- (i) We propose a novel PvCT framework and develop a set of verification algorithms that leverage our well-designed ADS to guarantee both integrity and

FIGURE 1: Information flow of publicly verifiable contact tracing.

accuracy of contact tracing result over both traditional travel records and transportation data.

- (ii) We provide rigorous security proof based on q -Strong Bilinear Diffie-Hellman assumption for the proposed publicly verifiable contact tracing algorithms.
- (iii) We perform detailed performance evaluation on three real-world datasets. Experimental results demonstrate that our algorithm is feasible with client CPU time in milliseconds and significantly reduces the storage overhead from the size of datasets to a constant 128 bytes.

The rest of the paper is organized as follows. Section 2 reviews existing studies on contact tracing and verifiable query processing. Section 3 formally defines the problem and its security model followed by cryptographic primitives and assumptions in Section 4. Section 5 presents the detailed PvCT algorithms based on a family of verifiable set accumulators. Security proof and performance evaluation are given in Sections 6 and 7, respectively. Finally, we conclude our paper in Section 8.

2. Related Work

2.1. Contact Tracing Algorithms. Due to the rapid spread of the COVID-19 pandemic and the importance of contact tracing, many research groups proposed their algorithms to improve contact tracing. Some of the algorithms rely on and expose records to a trusted third-party, such as BlueTrace [6], and some of them introduce a decentralized/public list approach: Private Kit [3] enables clients to log their own information (like locations) and can help the authority to contain an epidemic outbreak, Apple and Google [18, 19] have made joint efforts that support privacy-preserving contact tracing by inferring linkages, and Epione [4] provides end-to-end privacy-preserving contact tracing, or privacy-sensitive protocols and mechanisms for contact tracing [20], in which all personal data is locally stored on the phone and it is voluntary for users to publish/upload the data. Although contact tracing has been intensively studied as mentioned above, there are still no existing works that take resource-constrained clients into consideration. In other words, none of these works can be applied in cloud computing scenario. Table 1 provides a comparison of

TABLE 1: Comparison of contact tracing algorithms.

Algorithm	Accuracy	Integrity	Verifiability	Client's storage cost
Private Kit [3]	Yes	No	No	$O(N)$
BlueTrace [6]	Yes	No	No	$O(N)$
Epione [4]	Yes	No	No	$O(\log(N))$
PyC [8]	Yes	No	No	$O(N)$
BeepTrace [7]	Yes	Yes	No	$O(N)$
PvCT	Yes	Yes	Yes	$O(1)$

different contact tracing algorithms with respect to accuracy/integrity properties, client's storage cost, and verifiability, and N is the total number of contact tracing records. All of which are important for verifiable contact tracing in cloud computing scenario.

2.2. Verifiable Query Processing. Plenty of verifiable query processing algorithms have been studied to ensure the integrity of query results against an untrusted service provider (such as [15–17, 21–23]). Most of the existing works focus on outsourced databases and there are two basic approaches: enabling general queries using arithmetic/Boolean circuit-based verifiable computation schemes (SNARKs) and enabling ad-hoc queries using an authenticated data structure (ADS). Constructing efficient SNARKs and optimizing their implementation is a very active area of research [10–12, 24–28]. Pinocchio [12] utilized quadratic arithmetic programs to support arbitrary computation but at a very high expenses and occasionally impractical overhead. Moreover, it is difficult to amortize its preprocessing computation overhead if conducted a new one for each program. To remedy this issue, lots of work have been proposed, such as, recently, Xie et al. [11] proposed a zero-knowledge proof system in which the preprocessing time is only dependent on the size of the related circuit and irrespective of the circuit type.

The ADS-based algorithm is more efficient compared to the above SNARKs as it tailored to ad-hoc queries. Our proposed algorithm belongs to this sort of algorithms. An ADS is a special data structure with additional

authentication properties. In most cases, it has the form of “additional authentication value + regular data structure” so that the computation of the corresponding regular data structure can be outsourced to an untrusted server and the client can check if the returned result is valid. And, there are two basic techniques which are commonly utilized to serve as an ADS: digital signature and Merkle Hash Tree (MHT). Digital signatures employed asymmetric cryptography to verify the authenticity of digital messages. To support verifiable queries, it requires the data owner to sign every data record through his secret key; meanwhile, the verifier (client) can use the owner’s public key to verify the authenticity of a value and the signature of the value. Hence, it cannot scale up to large datasets [17]. MHT, on the other hand, demands only one signature on the root node of a hierarchical tree [29]. Each entry in a leaf node is assigned a hash digest of a data record, and each entry in an internal node is assigned a digest derived from the child nodes. The data owner signs the root of MHT, which can be used to verify any subset of data records. MHT has been widely adapted to various index structures, such as the authenticated prefix tree for multi-resource datasets [16] and the Merkle B-tree for relational data [21]. However, so far, no work has considered the integrity issue for verifiable contact tracing over cloud computing.

3. Problem Definition

Different from common settings such as [4, 5] in which the clients only require a returned result no matter whether it is authentic or not, we present the definition of the problem in publicly verifiable contact tracing setting as follows. The clients in our system not only submit their queries and expect to receive a result but also require that the correctness of the result can be verified. Besides, the service in our scenario is provided by two cloud service providers.

As shown in Figure 2, we assume the complete version of data which can be used in contact tracing is collected from each individual of a client (like students of the college), and it is stored in a database DB_p of a public cloud server (such as AWS or Azure). The records of the i th individual can be modeled as a set P_i , and $P_i = \{l_{ij} \| t_{ij} \| id_{ij} \mid j \in [m]\}$, where we assume that l_{ij} refers to locations where the i th individual had been to, t_{ij} is the related timestamp when the i th individual had been to that location, id_{ij} is the identity number of each individual which is known as a unique number of all the individuals belonging to the client, and m is total number of the records of i th individual.

Unlike traditional query scenario which only keeps one database, in our scenario, to protect the privacy of a diagnosed people as much as possible, we maintain a separate database called DB_s to keep the data of diagnosed people. In reality, this means that the related data of the diagnosed group is kept and obtained only through authorities, such as the hospital, Center of Disease Control, and other government departments. To support public query and verification, once there is a person who is diagnosed, the relevant authority will upload the patient’s location and related timestamps $D_i = \langle l_i \| t_i \rangle$ into a separate database DB_s which

can be hold in a different cloud server. Here, it should be noted that since identity number id_i is not essential for the final contact tracing matching part, DB_s only stores location and related timestamp of each diagnosed person. Meanwhile, in this way, we can shuffle the records of all the diagnosed people so that not only can it protect a diagnosed person’s personal information without any additional computation-intensive overhead (such as encryption) but also can it prevent any adversary from finding out whom the records belong to.

Notably, our contact tracing scheme is the only one that takes into consideration information beyond just location, which differentiates us from all existing ones. Considering that if an individual takes some transportation such as airplanes, trains, taxis, or buses, then, clearly, information about the transportation is of great importance. Therefore, the relevant transportation information of the i th individual (such as flight number, train number, and stations) is stored in both the databases. To enable verifiable contact tracing processing, an authenticated data structure (ADS) is constructed and embedded into each set of records uploaded by the clients or authorities.

3.1. System Model. Here, we now give a detailed description of our publicly verifiable contact tracing algorithm as follows.

First, in our scenario, clients are organizations such as colleges as we described in our introduction, and they would like to do check intimate contacts of their staffs under the rapid epidemic circumstances. Then, there are four main parties in this situation: individual client, authorities, Cloud Service Provider I (CSP_I), and Cloud Service Provider II (CSP_{II}). On this condition, we can easily find out there are two phases during the whole contact tracing processing: the query phase with CSP_I and matching phase with CSP_{II} . We discuss the following cases that may happen in these two phases separately.

Phase 1. In the query phase, clients may wish to search the records appearing in a certain time period uploaded by his choice. Specifically, query Q is in the form of $Q = ([t_b, t_s], id_i)$, where $[t_b, t_s]$ is a certain time window selected by the client and id is the identity number belonging to the client. As a result, CSP_I returns all records such that $\{re_i = \langle l_i \| t_i \| id_i \rangle \mid t_i \in [t_b, t_s] \wedge id_i = id\}$.

Example 1. In a COVID-19 contact tracing process, the time period of the query in Phase I can be a 14-day time window from the day he issued the query back to the beginning. Then, a client may issue a query $q = ([2020 - 10 - 01, 2020 - 10 - 14], Alice)$ to find all of the records stored in DB_p from October 1st to October 14th of 2020 and being associated with person whose ID is Alice.

Phase 2. After the query phase, in the matching phase, clients may want to utilize their staff’s records which are obtained in the query phase to find out whether they are positive contacts of diagnosed people or not. Then, a client

FIGURE 2: System model of PvCT.

may transfer his staff's records re_i s to CSP_{II} for executing the matching process. As a result, if the intersection I between the staff's records and the diagnosed people's records is empty, CSP_{II} returns the negative contact result to the client. Otherwise, CSP_{II} returns the positive contact result and the intersection to the client.

Example 2. Assume the target staff's records of clients are $\{2020/10/01 \parallel X \text{ shopping mall}, 2020/10/02 \parallel Y \text{ restaurant}, 2020/10/14 \parallel Z \text{ district}\}$, and the records of diagnosed people are $\{2020/10/01 \parallel A \text{ market}, 2020/10/03 \parallel B \text{ restaurant}, 2020/10/13 \parallel C \text{ district}\}$; then, after the clients send their staff's records to CSP_{II} to do the matching process, CSP_{II} can find out none of the diagnosed people had been to the same place as the staff, as the intersection between them is empty and the staff is a negative contact. Otherwise, if the records of diagnosed people are $\{2020/10/01 \parallel A \text{ market}, 2020/10/03 \parallel B \text{ restaurant}, 2020/10/130 \parallel C \text{ district} 2020/10/14 \parallel Z \text{ district}\}$, then the CSP_{II} can find out the staff is a positive one.

Additional examples can be found in Figure 2.

3.2. Threat Model. We consider CSPs, the two untrusted cloud service providers in the contact tracing framework, to be the potential adversary. Due to various issues such as security vulnerabilities, program bugs, and commercial interests, the CSPs can provide unfaithful contact tracing process, thereby returning incomplete or incorrect query and matching results. To address such kind of threat, we introduce publicly verifiable contact tracing that enable CSPs to prove the integrity and accuracy of query and matching results. Specifically, during the query phase, CSPs examines the ADS embedded in the records and constructs a verification object (VO) that includes the verification information of the related results. Using the VO, the client can establish the accuracy and integrity of the query and matching results, under the following criteria:

- (i) **Accuracy:** none of the records returned as results have been tampered with and all of them satisfy the query conditions. Meanwhile, there is no false positive/negative matching result can be verified correctly with nonnegligible possibility.
- (ii) **Integrity:** no valid record is missing regarding the query, and no positive result is missing in the matching phase.

Definition 1. (Accuracy). The result of public verifiable contact tracing is accurate if, for all PPT adversaries $\mathcal{A}$, there is a negligible function $\varepsilon(\cdot)$ such that

$$\Pr \left[\begin{array}{l} \left(\text{Verify } Q(\text{pk}, \text{acc}_{DB_p}, R, \pi_{x_i}, x_i) = 1 \wedge x_i \notin \text{Re} \vee \right) \\ \text{Verify } Q(\text{pk}, \text{acc}_{DB_p}, R, \pi_{x_i}, x_i) = 0 \wedge x_i \in \text{Re} \wedge \\ \left(\text{Verify } Q(\text{pk}, \text{acc}_{DB_s}, R, \pi_{\text{Re}}, \text{Re}) = 1 \wedge \text{negative} \vee \right) \\ \text{Verify } Q(\text{pk}, \text{acc}_{DB_s}, R, \pi_{\text{Re}}, \text{Re}) = 0 \wedge \text{positive} \end{array} \right] \leq \varepsilon(k). \quad (1)$$

The main challenge in this model is how to design an ADS which can be easily adapted to the contact tracing framework; meanwhile, VOs can be efficiently constructed, incurring small bandwidth overhead and fast verification time. We address this challenge in the next few sections.

4. Preliminaries

This section introduces major notations, as shown in Table 2, cryptographic primitives, and security assumptions that are used in our algorithms' design.

4.1. Cryptographic Primitives and Security Assumptions

4.1.1. Bilinear Pairings. Let $\mathbb{G}$ be a cyclic multiplicative group of prime order p and let g be a random generator of $\mathbb{G}$. $\mathbb{G}_T$ is also a cyclic multiplicative group of prime order p . Then, a bilinear pairing is a map $e: \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$, and the map e satisfies the following conditions:

- (i) **Bilinearity:** $e(g^a, g^b) = e(g, g)^{ab} = e(g^b, g^a), \forall a, b \in \mathbb{Z}_p$
- (ii) **Nondegeneracy:** there exists $e(g, g) \neq 1$, i.e., $e(g, g)$ generates $\mathbb{G}_T$
- (iii) **Computability:** group operations of $\mathbb{G}$ and the calculation of bilinear map e are both efficient, i.e., computable in polynomial time

For clarity of presentation, we assume, for the rest of the paper, a symmetric (Type I) pairing e . We note that our construction can be securely implemented in the (more efficient) asymmetric (Type III) pairing case, with straightforward modifications (refer to [30], for a general discussion on pairings). And, our security proof is based on the

TABLE 2: Major notations in our algorithm.

Symbol	Description
DB_*	Dataset of records
l_i	Location information
$date_j$	Date of j th record
Num	Dataset consists of all the num
pk	Public key
$\mathbb{G}, \mathbb{G}_T$	Cyclic multiplicative group
AccDigest	Accumulation value
$\parallel$	String concatenation
π	Proof of corresponding result
t_i	Timestamp of l_i
num	Number of records
Re	Query result dataset
sk	Secret key
g	Generator of group
κ	Security parameter
$[n]$	The set of integers $\{1, \dots, n\}$

q -Strong Bilinear Diffie–Hellman (q -SBDH) assumption over groups with bilinear pairings presented in [31].

Assumption 1. (q -Strong Bilinear Diffie–Hellman assumption). Let κ be the security parameter and $\text{pub} = (p, \mathbb{G}, \mathbb{G}_T, e, g)$ be a tuple of bilinear pairing parameters. For any probabilistic polynomial-time (PPT) adversary Adv and for q being a parameter of size polynomial in κ , there exists negligible probability $\text{neg}(\kappa)$ such that the following holds:

$$\Pr \left[\begin{array}{l} s \xleftarrow{\$} \mathbb{Z}_p^*, \gamma = e(g, g)^{1/(a+s)} \\ (a, \gamma) \in \mathbb{Z}_p^* \times \mathbb{G}_T \leftarrow \text{Adv}(\text{pub}, (g^s, \dots, g^{s^q})) \end{array} \right] \leq \text{neg}(\kappa). \quad (2)$$

Lemma 1 (see [32]). *The intersection of two sets X_1 and X_2 is empty if and only if there exist polynomials $\Phi_1(s)$ and $\Phi_2(s)$ such that $\Phi_1(s)\Omega(X_1) + \Phi_2(s)\Omega(X_2) = 1$, where $\Omega(X) = \prod_{x_i \in X} (x_i + s)$.*

The above result is based on extended Euclidean algorithms over polynomials and provides our essential verification process with the ability to check the correctness of empty set intersection.

4.1.2. Cryptographic Hash Function. A cryptographic hash function $\text{Hash}(\cdot)$ is a mathematical algorithm which takes an arbitrary length string as its input and returns a fixed-length bit string. It is a one-way function, i.e., a function which is practically infeasible to invert. Meanwhile, it is collision resistant meaning that it is computationally infeasible to find two different messages, m_1 and m_2 , such that $\text{Hash}(m_1) = \text{Hash}(m_2)$. Classic cryptographic hash functions include MD5, SHA-1, SHA-2, and SHA-3, where in recent years, the widely used hash function SHA-256 is a kind of SHA-2 family.

Lemma 2 (polynomial interpolation with FFT). *Assume there is a degree- n polynomial $\prod_{i=1}^n (x_i + s) = \sum_{i=0}^n b_i s^i$, and*

$b_n \neq 0, b_{n-1}, \dots, b_0$ is all the coefficients of the polynomial, and given $x_1, \dots, x_n$, it can be computed with $O(n \log n)$ complexity.

Lemma 2 presents an efficient process, given $x_1, \dots, x_n$, and the coefficients of a degree- n polynomial can be quickly computed. This lemma is based on an FFT algorithm [33] that computes the DFT in a finite field, such as $\mathbb{Z}_p$, and we used it in our constructions for arbitrary n and performing $O(n \log n)$ field operations. And, a detailed proof has been shown in [32], so we omit it here.

4.2. Cryptographic Set Accumulators. Our set accumulator is parameterized by a set of operations Q . For example, for our construction, it includes

- (1) subset $\subseteq$, intersection $\cap$: these functions take two sets as input and output a set. And, for intersection $\cap$, there exists two kinds of situations, and the first one is that the intersection is empty and the second one is that the intersection is not empty in which we should take its completeness into consideration.
- (2) $\in, \notin$: these functions take the set as input and output a value with type Boolean or integer (the output can also be viewed as a set with one element). Our set accumulators are all based on bilinear pairing and q -SBDH assumption as we have presented above.

Inspired by [32, 34, 35], we give formal definition of our set accumulators which consists of the following PPT algorithms:

- (i) $(\text{sk}, \text{pk}) \leftarrow \text{KeyGen}(1^\kappa)$: on input of the security parameter κ , it outputs a secret key sk and a public key pk .
- (ii) $\text{acc}(X) \leftarrow \text{Setup}(X, \text{pk})$: for a set X , it computes the accumulation value $\text{acc}(X)$ of X . In our construction, it can be efficiently computed without knowing the secret key sk and using pk only.
- (iii) $(\pi_Q, \mathcal{R}) \leftarrow \text{Proof } Q(X_1, X_2, Q, \text{pk})$: in input of a query Q , sets X_1, X_2 , and the public key pk , it returns the result $\mathcal{R} = Q(X_1, X_2)$ along with a proof π_Q .
- (iv) $b \leftarrow \text{Verify } Q(\text{acc}(X_1), \text{acc}(X_2), \mathcal{R}, \pi_Q, \text{pk})$: on input accumulation value $(\text{acc}(X_1), \text{acc}(X_2))$ of set X_1, X_2 , an result $\mathcal{R}$ and a proof π_Q for the query Q and public key pk , and it outputs b . If $b = 1$, the verification process indicates that the query result is valid; otherwise, the returned result is invalid and the accuracy and integrity of this query cannot be guaranteed.

More elaborated constructions of the set accumulator will be given in Section 5.3.

5. Constructions

In the following, Section 5.1 introduces the whole contact tracing process with Case I and Case II as we present in

Section 3. Then, we enrich our algorithm by taking transportation data into consideration (Section 5.2). Furthermore, we give the detailed constructions of our main cryptographic building block and set accumulator in Section 5.3.

5.1. ADS Construction and Verifiable Contact Tracing Process.

For simplicity, this section will only takes a client's contact tracing query Q which is over one individual into consideration. We assume that database DB_p stores all the people's data $P = \{P_1, P_2, \dots, P_n\}$, where n is total number of the people. The database DB_s stores all the diagnosed people's records $D = \{d_1, d_2, \dots, d_k\}$, where k is total number of the diagnosed people.

Recall that, in the proposed framework, an ADS is generated for every individual of each client (to be noted, the length of time period which is recorded for an individual depends on the type of the contagious disease, i.e., the incubation period of the disease decides the length of each period; in COVID-19, the period should be at least 14 days).

It can be utilized by the cloud service provider (CSP_I and CSP_{II}) to construct a verification object (VO) for each query. To this end, we extend the traditional data structure by adding an extra field, called *AccDigest*.

Moreover, *AccDigest* should have the following three desired properties to be functional as an ADS. First of all, *AccDigest* should be able to summarize an individual's records in a way that it can be used to construct a proof whether the result matches a query or not. Secondly, *AccDigest* should be able to support batching or aggregation verification of several devices of one individual or among different individuals. Thirdly, *AccDigest* should be in a constant size rather than a varying size grown in proportion to the number of records of an individual. Therefore, we propose to use accumulator as *AccDigest*:

$$\text{AccDigest} = \text{acc}(X) = \text{Setup}(X, \text{pk}), \quad (3)$$

where X stands for the target set which we would like to aggregate.

While for better readability, we defer detailed constructions to Section 5.3.

5.1.1. Verifiable Contact Tracing. Given a query Q by a client and two database DB_p and DB_s , at the end, the client needs to know the result is positive or negative. However, to be noted, in our scenario, recall the process of contact tracing, which we present in Section 3; there are actually two phases contained in a verifiable contact tracing processing, and we will discuss them separately.

5.1.2. Query Phase. The first phase is *Query Phase*. Assume all records over one staff of the client (like one of college's student) are $DB_c = \{c_i = l_i \| t_i \| \text{id}_i, i \in [n]\}$, where n is the number of records. Before the client uploads all the records to CSP_I , for the ease of generation of *AccDigest* and privacy concern, he utilizes a collision-free hash function *Hash* to hash every record into a fixed-length value, i.e.,

$H_{DB_c} = \text{Hash}(DB_c) = \{\text{Hash}(c_i), i \in [n]\}$. Then, the client generates an accumulation value $\text{AccDigest}_c = \text{Setup}(H_{DB_c}, \text{pk})$ over the set of all his hashed records. Meanwhile, the client introduces a counter to count how many records the device collected each day; for the record set of the j th date date_j is $D_{\text{date},j}$, the device has $\text{num}_j = \text{count}(D_{\text{date},j})$ records; then, we store it as an additional information for our verification. If there are records of t days that the device will upload to CSP_I , then we set $\text{Num} = \{\text{num}_j \| \text{date}_j | j \in [t]\}$.

After all the above setup processes, the client can issue a query $Q = \langle t_i, \text{id}_i \rangle$ to CSP_I to obtain all of his records. Here, t_i is the date that the client would like to retrieve his records and id_i is the identity number of an individual which the client wants to retrieve over this query. Obviously, *the main challenge in this phase is how to verify both the correctness and integrity of the returned result using the corresponding AccDigest*.

For instance, the query condition which the client issued is $Q = \langle 2020/9/1 - 2020/9/15, \text{Alice} \rangle$, that is to say, the result that the client would like to obtain all his records over an individual and his ID is *Alice*, and the date the client issued is from 2020.09.01 to 2020.09.15. Because our verification algorithm is independent of the retrieval algorithm, in other words, any existing retrieval algorithm such as [36] can be used in our construction and do not affect accuracy and security of our algorithm, we omit the detail of retrieval process here and only consider after CSP_I finished its retrieval process and obtain the corresponding result Re . Then, we can apply $\text{Setup}(\text{Re}, \text{pk})$ to generate a proof π and use a counter to obtain the number of result set num_{re} as the VO for the retrieval result. Accordingly, the client can first check if the number of records in the returned result set Re is equal to the sum of numbers $\sum_{j \in [\text{date}_c, \text{date}_e]} \text{num}_j$ stored in the client side (from the initial day to the last day of the query Q). If the check holds, then the client can utilize $e(\text{AccDigest}_c, \pi_1) \stackrel{?}{=} e(\pi_2, g)$ to verify the accuracy of the result Re . The whole process of this phase is detailedly specified in Algorithm 1. If the verification process in this phase fails, i.e., $b = 0$, then the whole contact tracing execution aborts. Otherwise, the verification process in the query phase succeeds, then we proceed to the matching phase.

5.1.3. Matching Phase. The second phase is the *Matching Phase*. In this phase, the client issues a matching query $Q = \text{Re}^*$ (Re^* here is a variation of set Re , due to id_i , i.e., the identity number of the result does not required in the matching process, $\text{Re}^* = \{l_i \| t_i, i \in [t]\}$, t is the number of records in the result set) to the CSP_{II} in order to find out whether he/she is a close contact to diagnosed people or not. From the view of mathematics, the problem which we try to solve here is to find out whether there is an intersection I between set Re^* and set $DB_s = \{d_i = l_i \| t_i, i \in [m]\}$, where m is the number of records over all the diagnosed people. Then, the main challenge in this phase for our scheme is how to verify whether the intersection I is empty or not. Recall what we described in Section 3, and the records of diagnosed people are owned by the authorities, such as hospitals or

```

ADS Generation (by the client)
for each record  $c_i = l_i || t_i || id_i, i \in [n]$  over an individual of the client do
   $H_{DB_c} = \{\text{Hash}(c_i), i \in [n]\}$ ;
end
 $\text{AccDigest}_c \leftarrow \text{Setup}(H_{DB_c}, \text{pk})$ ;
 $\text{Num} = \{\text{num}_j || \text{date}_j | j \in [t]\}$ ;
Store  $\text{AccDigest}_c, \text{Num}$  in the client side;
VO Construction (by the CSPI)
Input: Query  $Q = \langle \text{date}, \text{id} \rangle$ 
if  $\text{Re} = \text{search}(Q, \text{DB}_p)$  then
   $\pi_1 \leftarrow \text{Proof Sub}(H_{\text{Re}}, \text{AccDigest}_c, \text{pk})$ ;
   $\pi_2 \leftarrow \text{Setup}(H_{\text{Re}}, \text{pk})$ ;
   $\pi = (\pi_1, \pi_2)$ ;
   $\text{num}_{\text{re}} = \text{count}(\text{Re})$ ;
  Send  $\langle \text{Re}, \pi, \text{num}_{\text{re}} \rangle$  to the client;
end
Result Verification (by the client)
if  $\text{num}_{\text{re}} = \sum_{j \in [\text{date}_s, \text{date}_e]} \text{num}_j$  then
  if  $e(\text{AccDigest}_c, \pi_1) = e(\pi_2, g)$  then
    Output:  $b = 1$ 
  else
    Output:  $b = 0$ 
  end
end
end

```

ALGORITHM 1: The query phase of verifiable contact tracing over an individual.

government. Then, before they upload these records to CSP_{II}, the authorities generate an accumulation value $\text{AccDigest}_{\text{au}} = \text{Setup}(H_{\text{DB}_s}, \text{pk})$ and make this value public to all the clients for further verification.

For example, if the result set that the client obtained is $\text{Re}^* = \{2020/09/01 || \text{A laboratory}, 2020/09/02 || \text{B market}, 2020/09/03 || \text{C restaurant}\}$. Assume there is an intersection $I = \text{Re}^* \cap \text{DB}_s$; then, if the intersection I is empty, it means that none of the diagnosed people have been to A laboratory, B market, and C restaurant at the same time with the client. This also means the client is a negative contact. Then, CSP_{II} can utilize $\text{Proof Emp}(H_{\text{Re}^*}, \text{DB}_s, \text{pk})$ to generate a proof π_1 and send the result $\langle \text{negative}, \pi \rangle$ to the client. According to the result, the client can use $\text{Verify Emp}(\text{acc}(\text{Re}^*), \text{AccDigest}_{\text{au}}, \pi_1, \text{pk})$ to verify if the negative judge by CSP_{II} is trustworthy. If $b = 1$, it means the verification process in this phase succeeds, and the client can be sure of the negative result. Otherwise, the client will refuse to believe the negative result.

At the meantime, if the intersection I is not empty, which means at least one of the diagnosed people has been to A lab, B park, or C restaurant at the same time with the client. We assume $I = \{2020/09/01 || \text{A laboratory}\}$; obviously, the client is a positive contact which is with high possibility to be infected. Then, CSP_{II} can apply $\text{Proof Int}(H_{\text{Re}^*}, H_{\text{DB}_s}, H_I, \text{pk})$ to generate a proof π and send the result $\langle \text{acc}(H_I), \text{positive}, \pi \rangle$ to the client. According to the result, the client can use $\text{Verify Int}(\text{acc}(H_{\text{Re}^*}), \text{AccDigest}_{\text{au}}, \text{acc}(H_I), \pi_2, \text{pk})$ to verify if the negative judge by CSP_{II} is trustworthy. If $b = 1$, it means the verification process in this phase succeeds, and the client can be sure of the positive result and he can just seek medical care as soon as possible.

Otherwise, the client will refuse to believe the positive result. The whole process of this phase is detailed in Algorithm 2.

5.2. Transportation Data. There is a special kind of data needing further discussion, which is transportation data. Assume clients or diagnosed people have used vehicles (such as airplanes, trains, buses, or taxis) for short or long trips; then, obviously, data of vehicles (such as flight/train number, bus plate, departure, and terminal station) is significant in our contact tracing algorithm. For example, if a diagnosed person has taken an airplane, then besides the departure and terminal station information, the closest contact of this diagnosed person is the passengers who took the same flight; in other words, the flight number is essential for the matching process. As for trains and other public transport such as buses or MRT, obviously there may exist multiple stations rather than an initial station and a terminal station. Therefore, the information of middle stations is also important in our contact tracing process. To conceptualize this, we analyze two possible cases of transportation dataset that may happen in the matching process, as shown in Figure 3.

In the first case shown in Figure 3(a), we assume that the client got on the bus at initial station and got off at middle station 1 and a diagnosed person got on the same bus at middle station i ($i > 1$). Then, mathematically, there exists two datasets consisting of station information; the first set starts from the initial station to middle station 1, i.e., $\text{Trans}_c = \{\text{Initial station}, \text{Middle Station 1}\}$; the other set starts from middle station i and ends at the terminal station of the bus, i.e., $\text{Tran}_d = \{\text{Middle Station } i, \text{Middle Station } \dots\}$.


```

ADS Generation (by the authorities)
for each records  $d_i = l_i \| t_i$ ,  $i \in [m]$  of diagnosed people do
   $H_{DB_s} = \{\text{Hash}(d_i), i \in [m]\}$ ;
end
 $\text{AccDigest}_{\text{au}} \leftarrow \text{Setup}(H_{DB_s}, \text{pk})$ ;
 $\text{AccDigest}_{\text{Tran}_d} \leftarrow \text{Setup}(H_{\text{Tran}_d}, \text{pk})$ ;
Public  $\text{AccDigest}_{\text{au}}, \text{AccDigest}_{\text{Tran}_d}$ ;
VO Construction (by the CSPIT)
Input:  $\text{Re}^*$ 
while
   $I = \text{Re}^* \cap DB_s$  and  $I^* = \text{Tran}_c \cap \text{Tran}_d$  do
    if  $I = \emptyset$  then
       $\pi_1 \leftarrow \text{Proof No M}(H_{\text{OffSta}}, H_{\text{Tran}_d}, \text{pk})$ ;
       $\pi_2 \leftarrow \text{Proof Emp}(H_{\text{Re}^*}, H_{DB_s}, \text{pk})$ ;
       $\pi = (\pi_1, \pi_2)$ ;
      Send  $\langle \text{negative}, \pi \rangle$  to the client;
    else
      if  $I^* \neq \emptyset$  then
         $\pi_1 \leftarrow \text{Proof Mem}(H_{\text{OffSta}}, H_{\text{Tran}_d}, \text{pk})$ ;
         $\pi_2 \leftarrow \text{Proof Int}(H_{\text{Re}^*}, H_{DB_s}, H_I, \text{pk})$ ;
         $\pi = (\pi_1, \pi_2)$ ;
        Send  $\langle \text{positive}, I, \text{OffSta}, \pi \rangle$  to the client;
      else
         $\pi \leftarrow \text{Proof Int}(H_{\text{Re}^*}, H_{DB_s}, H_I, \text{pk})$ ;
        Send  $\langle \text{acc}(H_I), \text{positive}, \pi \rangle$  to the client;
      end
    end
  end
end
Result Verification (by the client)
if negative then
   $b_1 \leftarrow \text{Verify No M}(\text{AccDigest}_{\text{Tran}_d}, H_{\text{OffSta}}, \pi_1, \text{pk})$ ;
   $b_2 \leftarrow \text{Verify Emp}(\text{acc}(H_{\text{Re}^*}), \text{AccDigest}_{\text{au}}, \pi_2, \text{pk})$ ;
   $b = \underline{b_1} \vee b_2$ ;
  Output  $b$ ;
else
   $b_1 \leftarrow \text{Verify Mem}(\text{AccDigest}_{\text{Tran}_d}, H_{\text{OffSta}}, \pi_1, \text{pk})$ ;
   $b_2 \leftarrow \text{Verify Int}(\text{acc}(H_{\text{Re}^*}), \text{AccDigest}_{\text{au}}, \text{acc}(H_I), \pi_2, \text{pk})$ ;
   $b = \underline{b_1} \vee b_2$ ;
  Output  $b$ ;
end

```

ALGORITHM 2: The matching phase of verifiable contact tracing over an individual.

$i + 1, \dots, \text{Terminal Station}\}$. Obviously, under this circumstance, there is no intersection between set Tran_c and set Tran_d , in other words, the client is not a positive contact.

In the second case shown in Figure 3(b), we suppose that the client got on the bus at initial station and got off at middle station i ($i > 1$); at the meantime, a diagnosed person got on the same bus at middle station 1. Same as Case 1, there exists two datasets; the first set Tran_c starts from the initial station to middle station i , and the other starts from middle station 1 to the terminal station. It is easy to find out there exists an intersection $I^* = \text{Tran}_c \cap \text{Tran}_d$, and this intersection I^* is not empty. That is to say, the client is a positive contact.

Based on the above analysis of different cases that may happened to transportation data, it requires us to do some additional precomputation on both the client and authorities sides. Meanwhile, in the VO construction part, it also needs to generate corresponding proof to transportation data.

First, for all circumstances, both the client and authorities need to generate accumulation value of their transportation data set through $\text{AccDigest}_{\text{Tran}} = \text{Setup}(\text{Tran}, \text{pk})$ for further verification. Meanwhile, it is easy to find out that there is no need to verify the transportation data separately in the query phase because the integrity check of the whole dataset can also pledge to the transportation data.

FIGURE 3: Case analysis of transportation data in matching process (a) case I and (b) case II.

Then, when we execute the matching process, as analyzed above, CSP_{II} has to check whether there is an intersection between the transportation dataset of the client and diagnosed people. As we described before, the first case is that $I^* = Tran_c \cap Tran_d$ is empty. Mathematically, we can easily find out that proving intersection of these two sets is empty which is equivalent to prove the off station $OffSta$ of the client is not a member of the transportation dataset of diagnosed people. We utilize $Proof\ No\ M(H_{OffSta}, H_{Tran_d}, pk)$ to generate a corresponding proof π^* , which will be sent to the client along with a negative contact tracing result. Accordingly, the client can use $Verify\ No\ M(AccDigest_{Tran_d}, H_{OffSta}, \pi_1, pk)$ to verify the negative result.

The second case is that $I^* = (Tran_c, Tran_d)$ is not empty. Similarly, we can easily find out proving intersection of these two sets is not empty, is equivalent to proving that the off station $OffSta$ is a member of the station set of diagnosed people $Tran_d$. Then, we can utilize $Proof\ Mem(H_{OffSta}, H_{Tran_d}, pk)$ to generate a corresponding proof π^* , which will be sent to the client along with a positive contact tracing result. Accordingly, the client can use $Verify\ Mem(AccDigest_{Tran_d}, H_{OffSta}, \pi_1, pk)$ to verify the positive result. All the detailed process is shown in bold print in Algorithms 1 and 2.

5.3. Construction of Set Accumulators. We now discuss the possible construction of the accumulator which can be used in Section 5.1.

Inspired by [32], we present a construction which is based on q -SBDH and bilinear pairing assumption. It consists of the following algorithms.

- (i) $(sk, pk) \leftarrow KeyGen(1^\kappa)$: let $(p, \mathbb{G}, \mathbb{G}_T, e, g)$ be a bilinear pairing. Randomly choose s from $\mathbb{Z}_q$. Then, it outputs a secret key $sk = s$ and $pk = \{g^{s^i} : i \in [q]\}$.
- (ii) $acc(X) \leftarrow Setup(X, pk)$: for a set $X = \{x_1, x_2, \dots, x_n\}$, its accumulation value $acc(X) = g^{\Omega(X)} = g^{\prod_{x_i \in X} (x_i + s)}$. Owing to the property of the

polynomial interpolation with FFT, it can be efficiently computed without knowing the secret key s .

To make a clear expression of our construction, here, we split the procedures that show how to construct four core proof and verify protocols that meet different query requirements into cases as follows.

5.3.1. Subset

- (i) $\pi_1 \leftarrow Proof\ Sub(X_1, X_2, pk)$: given two sets (X_1, X_2) and public key pk , to verify whether X_1 is a subset of X_2 , i.e., $X_1 \subseteq X_2$, we can compute $\pi_1 = acc(X_2 - X_1) = g^{\prod_{x_j \in X_2 - X_1} (x_j + s)}$.
- (ii) $b \leftarrow Verify\ Sub(acc(X_1), acc(X_2), \pi_1, pk)$: the client verifies the following equation:

$$e(acc(X_1), \pi_1) \stackrel{?}{=} e(acc(X_2), g), \quad (4)$$

This equation holds if and only if X_1 is a subset of X_2 . In other words, if π_1 is verified as correct, the client is assured that $X_1 \subseteq X_2$; then, output $b = 1$. Else, output $b = 0$.

5.3.2. Empty

- (i) $\pi_2 \leftarrow Proof\ Emp(X_1, X_2, pk)$: given two sets (X_1, X_2) and public key pk , it verifies whether $X_1 \cap X_2 = \emptyset$. Based on the extended Euclidean algorithms over polynomials, if and only if $X_1 \cap X_2 = \emptyset$, there exists polynomials $\Phi_1(s)$ and $\Phi_2(s)$ such that $\Phi_1(s)\Omega(X_1) + \Phi_2(s)\Omega(X_2) = 1$. In that case, we can compute $\pi_2 = (\pi_{21}, \pi_{22}) = (g^{\Phi_1(s)}, g^{\Phi_2(s)})$.
- (ii) $b \leftarrow Verify\ Emp(acc(X_1), acc(X_2), \pi_2, pk)$: the client verifies the following equation:

$$e(acc(X_1), \pi_{21}) \cdot e(acc(X_2), \pi_{22}) \stackrel{?}{=} e(g, g). \quad (5)$$

And, this equation holds if and only if the intersection of X_1 and X_2 is empty. That is to say, if π_2 is verified as correct, the client is assured that $X_1 \cap X_2 = \emptyset$. Then, output $b = 1$, else output $b = 0$.

5.3.3. Completeness

- (i) $\pi_3 \leftarrow \text{Proof Com}(X_1, X_2, I, \text{pk})$: let I be the intersection of X_1 and X_2 , i.e., $I = X_1 \cap X_2$. Similar to computing the proof of empty, based on the extended Euclidean algorithms over polynomials, if and only if $(X_1 - I) \cap (X_2 - I) = \emptyset$, there exists polynomials $\Phi_3(s)$ and $\Phi_4(s)$ such that $\Phi_3(s)\Omega(X_1 - I) + \Phi_4(s)\Omega(X_2 - I) = 1$. In that case, we can compute $\pi_3 = (\pi_{31}, \pi_{32}) = (g^{\Phi_3(s)}, g^{\Phi_4(s)})$.
- (ii) $b \leftarrow \text{Verify Com}(\text{acc}(X_1 - I), \text{acc}(X_2 - I), I, \pi_3, \text{pk})$: the client verifies the proof π_3 through the following equation:

$$e(\text{acc}(X_1 - I), \pi_{31}) \cdot e(\text{acc}(X_2 - I), \pi_{32}) \stackrel{?}{=} e(g, g). \quad (6)$$

And, this equation holds if and only if the intersection of $X_1 - I$ and $X_2 - I$ is empty. In other words, if π_3 is verified as correct, the client is assured that set I contains *all* the common elements between X_1 and X_2 . Then, output $b = 1$, else output $b = 0$.

5.3.4. Intersection

- (i) $\pi_4 \leftarrow \text{Proof Int}(X_1, X_2, I, \text{pk})$: let I be the intersection of X_1 and X_2 . Obviously, the correctness of the set intersection operation can be expressed by the combination of subset and completeness condition. That means $I = X_1 \cap X_2$ holds, if and only if the following two conditions holds:

- (1) $I \subseteq X_1 \wedge I \subseteq X_2$
- (2) $(X_1 - I) \cap (X_2 - I) = \emptyset$

According to the conditions mentioned above, we can easily find out that

$$\begin{aligned} \pi_4 &= (\pi_{41}, \pi_{42}, \pi_{43}) \\ &= (\text{Proof Sub}(I, X_1, \text{pk}), \text{Proof Sub}(I, X_2, \text{pk}), \\ &\quad \text{Proof Com}(X_1, X_2, I, \text{pk})). \end{aligned} \quad (7)$$

- (ii) $b \leftarrow \text{Verify Int}(\text{acc}(X_1), \text{acc}(X_2), \text{acc}(I), \pi_4, \text{pk})$: first, the client verifies the subset condition by checking the following equation:

$$e(\text{acc}(I), \pi_{4i}) \stackrel{?}{=} e(\text{acc}(X_i), g), \quad \text{for } i = 1, 2. \quad (8)$$

If the above check on subset proof succeeds, the client verifies the completeness condition through checking the following equation:

$$e(\pi_{41}, \pi_{43}) \cdot e(\pi_{42}, \pi_{43}) \stackrel{?}{=} e(g, g). \quad (9)$$

If the above equation holds, then the client is assured that I is the correct intersection. Then, output $b = 1$, else output $b = 0$.

5.3.5. Membership

- (i) $\pi_5 \leftarrow \text{Proof Mem}(x_i, X, \text{pk})$: let x_i be an element of set X , i.e., $x_i \in X$. Then, we can compute $\pi_5 = \text{acc}(X/x_i) = g^{\prod_{x_j \in X \setminus \{x_i\}} (x_j + s)}$.
- (ii) $b \leftarrow \text{Verify Mem}(\text{acc}(X), x_i, \pi_5, \text{pk})$: the client verifies the membership using the following equation:

$$e(\text{acc}(X), g) \stackrel{?}{=} e(\pi_5, g^{x_i}). \quad (10)$$

If the verification succeeds, output $b = 1$; the client is assured that x_i is an element of set X . Else, output $b = 0$.

5.3.6. Nonmembership

- (i) $\pi_6 \leftarrow \text{Proof No}(y, X, \text{pk})$: let y be an element which is not belong to set X , i.e., $y \notin X$. Set $\pi_{61} = -\Omega(-y) \bmod p = -\prod_{x_i \in X} (x_i - y) \bmod p$; then, accordingly, we can compute $\pi_{62} = g^{(\Omega(s) - \Omega(-y)/y + s)}$, and set $\pi_6 = (\pi_{61}, \pi_{62})$.
- (ii) $b \leftarrow \text{Verify No } M(\text{acc}(X), y, \pi_6, \text{pk})$: the client can verify the nonmembership using the following equation:

$$e(\text{acc}(X) \cdot g^{\pi_{61}}, g) \stackrel{?}{=} e(\pi_{62}, g^{y+s}). \quad (11)$$

If the verification succeeds, output $b = 1$; the client is assured that y is not an element of set X . Else output $b = 0$.

6. Security Proof

In this section, we provide security proofs for our schemes; more specifically, proofs of security for the six set-related operations: *Subset* (), *Empty* (), *Completeness* (), *Intersection* (), *Membership* (), and *Nonmembership* (), in accumulator settings. We will first provide security proofs for two more fundamental set-related operations, namely, *Set Containment* and *Set Disjointness*, and then, reduce the security of the six set-related operations in our scheme to the two.

- (1) *Set Containment*(input₁, input₂): this operation takes a set X_1 or an element x belonging to the universe as its first input, and a set X_2 as its second input. It outputs “1” if $X_1 \subseteq X_2$ or $x \in X_1$ and outputs “0” otherwise.

It is a generalization of *Subset*() and *Membership*(), which provides a unified interface for the two. Informally speaking, if one wants to check whether a set X_1 is a subset of X_2 or whether x is an element of X_2 , in both cases she can use set containment operation. If the inputs are two sets, it is equivalent to

Subset(), and if the inputs are an element and a set, it is equivalent to *Membership()*.

- (2) *Set Disjointness* ($X_1, X_2, \dots, X_k$): this operation takes as inputs k ($k \geq 2$) sets $X_1, X_2, \dots, X_k$, and it outputs “1” if $X_1 \cap X_2 \cap \dots \cap X_k = \emptyset$, outputs “0” otherwise.

We then proceed to prove that if there exists an adversary $\mathcal{A}$ who is able to create legal witness for an incorrect set operation result, an algorithm can be constructed to break *q-Strong Bilinear Diffie-Hellman assumption*. We define our security games at first.

6.1. Security Game 1. *q-Strong Bilinear Diffie-Hellman Game*: in this game, an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ involve in an interactive process:

- (1) $\mathcal{C}$ prepares a *q-SBDH* instance $\text{ins} = \{(p, \mathbb{G}, \mathbb{G}_T, e, g), g^s, \dots, g^{s^q}\}$ and sends it to $\mathcal{A}$
- (2) If $\mathcal{A}$ can return a legal pair $(a, e(g, g)^{(1/a+s)})$, we say that $\mathcal{A}$ wins this game, and thus, it breaks the *q-SBDH* assumption

6.2. Security Game 2. *Valid Witness for Incorrect Set Operation Result Game*: this is an abstraction of security games for all the related set operations in our scheme. So, in the description of this game, we do not refer to any concrete set operation:

- (1) $\mathcal{C}$ prepares a group of system parameters $\text{params} = \{(p, \mathbb{G}, \mathbb{G}_T, e, g), \text{sk} = s, \text{pk} = \{g^{s^i} : i \in [q]\}\}$ (even if it sends the public part of params to adversary $\mathcal{A}$, there is no essential difference except that in fact $\mathcal{A}$ does not need to query $\mathcal{C}$ for set witness; it is similar to security games in general public-key encryption, where adversary $\mathcal{A}$ can encrypt message itself as well as query and get response from challenger $\mathcal{C}$).
- (2) $\mathcal{A}$ issues witness query on the arbitrary set X she wants, subject to which the cardinality of X should be less than or equal to q . The total query number is also bounded by a polynomial of security parameter, to which we only implicitly refer.
- (3) After the query phase, if $\mathcal{A}$ can return a legal witness (or several legal witnesses) for an incorrect set operation result, we say that $\mathcal{A}$ wins this game, and thus, it breaks security of our scheme.

Theorem 1. *If there exists an adversary who can provide a valid witness for an incorrect set containment operation result, there exists another algorithm can break *q-Strong Bilinear Diffie-Hellman assumption*.*

Let $\text{pub} = (p, \mathbb{G}, \mathbb{G}_T, e, g)$ be a tuple of bilinear pairing parameters. Given elements $g, g^s, \dots, g^{s^q} \in \mathbb{G}$, while s is chosen uniformly at random from $\mathbb{Z}_p^*$, suppose there exists a polynomial-time algorithm $\mathcal{A}$ that can find two sets X_1 and X_2 and a legal witness W such that $X_1 \not\subseteq X_2$ and $e(\text{acc}(X_1), W) = e(\text{acc}(X_2), g)$. Then, we can use $\mathcal{A}$ to

construct a polynomial-time algorithm $\mathcal{A}'$ to break *q-Strong Bilinear Diffie-Hellman assumption*.

Proof. The main idea behind the proof is that algorithm $\mathcal{A}'$ simultaneously takes part in two security games and sits between challenger $\mathcal{C}$ (in *q-Strong Bilinear Diffie-Hellman* security game) and algorithm $\mathcal{A}$ (in *Set Containment* security game). It can prepare parameters for $\mathcal{A}$ by utilizing what it receives from $\mathcal{C}$, and then, it forms its own solution to a *q-Strong Bilinear Diffie-Hellman* instance after some calculation of $\mathcal{A}$'s response:

- (1) Algorithm $\mathcal{A}'$ first interacts with challenger $\mathcal{C}$. It will receive a *q-Strong Bilinear Diffie-Hellman* instance to be challenged upon. W.l.o.g., and we denote this instance as $\text{ins} = \{(p, \mathbb{G}, \mathbb{G}_T, e, g), g^s, \dots, g^{s^q}\}$. And, if it can successfully find a pair $(a, e(g, g)^{(1/a+s)})$, we say that it succeeds in breaking the *q-Strong Bilinear Diffie-Hellman* assumption.
- (2) Algorithm $\mathcal{A}$ can arbitrarily choose a set X it wants as a query, with the only restriction that the cardinality of that set cannot be larger than q . Suppose it chooses X , sends it to algorithm $\mathcal{A}'$, and asks for the accumulation value for it.
- (3) With parameters in ins , algorithm $\mathcal{A}'$ can easily respond to the request from $\mathcal{A}$ in the last step. For example, to generate $\text{acc}(X)$ for a set X , $\mathcal{A}'$ first calculates all the coefficients of polynomial $\prod_{x_i \in X} (s + x_i)$, denoted as $\alpha_0, \alpha_1, \dots, \alpha_m \in \mathbb{Z}_p$. Then, it calculates $\text{acc}(X) = g^{\alpha_0} \times (g^s)^{\alpha_1} \times \dots \times (g^{s^m})^{\alpha_m} = g^{\sum \alpha_i s^i} = g^{\sum_{x_i \in X} (s + x_i)}$. After the calculation, it sends $\text{acc}(X)$ as the answer to $\mathcal{A}$.
- (4) $\mathcal{A}$ may conduct other queries (notice that element update operation can be included in this case, e.g., two queries for set $X_1 = \{x_1, x_2\}$ and set $X_2 = \{x_1, x_2, x_3\}$ are equivalent to one query for an update (with value insertion) on X_1 , and identical to a delete operation on X_2 as well.) for the sets it wants, and $\mathcal{A}'$ will respond to it accordingly, which subjects to the condition that there is a upper bound of the total query number.
- (5) After the query phase, $\mathcal{A}$ generates two pairs $(X_1, \text{acc}(X_1))$ and $(X_2, \text{acc}(X_2))$, where $X_1 = \{x_{1,1}, x_{1,2}, \dots, x_{1,m}\}$ and $X_2 = \{x_{2,1}, x_{2,2}, \dots, x_{2,n}\}$ are the two sets, $\text{acc}(X_1) = g^{\sum_{x_{1,i} \in X_1} (s + x_{1,i})}$ and $\text{acc}(X_2) = g^{\sum_{x_{2,j} \in X_2} (s + x_{2,j})}$, are the corresponding accumulation values. It also generates a legal witness W such that $e(\text{acc}(X_1), W) = e(\text{acc}(X_2), g)$, but there is at least one element $\hat{x}$ satisfies that $\hat{x} \in X_1$ and $\hat{x} \notin X_2$. Then, it sends these values back to $\mathcal{A}'$.
- (6) $\mathcal{A}'$ sends to challenger $\mathcal{C}$ a pair $\text{chl} = (\hat{x}, [e(g, W)^{P(s)} \cdot e(g, g)^{-Q(s)}]^{R^{-1}})$, where $P(s) =$, $Q(s) =$, and R is a zero-order polynomial, i.e., a constant number. And, it breaks *q-Strong Bilinear Diffie-Hellman* assumption.

The reason why chl in Step 6 is a successful q -SBDH pair is that

$$\begin{aligned} e(\text{acc}(X_1), W) &= e(\text{acc}(X_2), g) \\ \Rightarrow e(g, W)^{x_{1,i} \in X_1} &= e(g, g)^{x_{2,j} \in X_2} \end{aligned} \quad (12)$$

Since $\prod_{x_{2,j} \in X_2} (s + x_{2,j})$ cannot be divided by $s + \hat{x}$, it is reasonable to assume that $\prod_{x_{2,j} \in X_2} (s + x_{2,j}) = Q(s) (s + \hat{x}) + R$. For ease of readability, we let $P(s) = \prod_{x_{1,i} \in X_1} (s + x_{1,i})/s + \hat{x}$. So, we have

$$\begin{aligned} e(g, W)^{P(s) * (s + \hat{x})} &= e(g, g)^{Q(s) * (s + \hat{x}) + R} \\ \Rightarrow e(g, W)^{P(s)} &= e(g, g)^{Q(s) + (R/s + \hat{x})} \\ \Rightarrow e(g, W)^{P(s)} \cdot e(g, g)^{-Q(s)} &= e(g, g)^{(R/s + \hat{x})} \\ \Rightarrow [e(g, W)^{P(s)} \cdot e(g, g)^{-Q(s)}]^{R^{-1}} &= e(g, g)^{(1/s + \hat{x})}. \end{aligned} \quad (13)$$

That is, chl = $(\hat{x}, [e(g, W)^{P(s)} \cdot e(g, g)^{-Q(s)}]^{R^{-1}})$ is a legal pair to break q -SBDH assumption. $\square$

Theorem 2. *If there exists an adversary who can provide a valid witness for an incorrect set disjointness operation result, there exists another algorithm can break q - Strong Bilinear Diffie-Hellman assumption.*

Let $\text{pub} = (p, \mathbb{G}, \mathbb{G}_T, e, g)$ be a tuple of bilinear pairing parameters. Given elements $g, g^s, \dots, g^{s^q} \in \mathbb{G}$, while s is chosen uniformly at random from $\mathbb{Z}_p^*$; suppose there exists a polynomial-time algorithm $\mathcal{A}$ that can find a group of k sets $X_1 = \{x_{1,1}, \dots\}, X_2 = \{x_{2,1}, \dots\}, \dots, X_k = \{x_{k,1}, \dots\}$, a group of k polynomials $q_1(s), \dots, q_k(s)$, and k pairs of legal witnesses $(W_i = g^{P_i(s)} = g^{\prod_{x \in X_i} (s + x)}, F_i = g^{q_i(s)})$ such that $X_1 \cap X_2 \cap \dots \cap X_k \neq \emptyset$ and $\prod_{1 \leq i \leq k} e(g^{P_i(s)}, g^{q_i(s)}) = e(g, g)$. Then, we can use $\mathcal{A}$ to construct a polynomial-time algorithm $\mathcal{A}'$ to break q - Strong Bilinear Diffie-Hellman assumption.

Proof. Recall steps in the proof of Theorem 1, compared to which the only difference in this proof is that, after the query phase, $\mathcal{A}$ returns to $\mathcal{A}'$ a bunch of different values: $\{X_i, q_i(s), W_i, F_i\}_{i \in [1, \dots, k]}$. So, we omit those steps and go straight to the deduction that why with those returned values $\mathcal{A}'$ can break q -SBDH assumption.

Since $X_1 \cap X_2 \cap \dots \cap X_k \neq \emptyset$, we assume that one of the common elements of these k sets is $\bar{x}$, and therefore, we introduce a new notation $P'_i(s) = (P_i(s)/s + \bar{x})$. Thus, we have the following deduction:

$$\begin{aligned} e(g, g) &= \prod_{1 \leq i \leq k} e(g^{P_i(s)}, g^{q_i(s)}) \\ &= \prod_{1 \leq i \leq k} e(g^{(s + \bar{x})P'_i(s)}, g^{q_i(s)}) \\ &= \left(\prod_{1 \leq i \leq k} e(g^{P'_i(s)}, g^{q_i(s)}) \right)^{s + \bar{x}} \\ \Rightarrow e(g, g)^{(1/s + \bar{x})} &= \prod_{1 \leq i \leq k} e(g^{P'_i(s)}, g^{q_i(s)}), \end{aligned} \quad (14)$$

That is, chl = $(\bar{x}, \prod_{1 \leq i \leq k} e(g^{P'_i(s)}, g^{q_i(s)}))$ is a legal pair to break q -SBDH assumption.

Based on Theorem 1 and Theorem 2, the security of our scheme is obvious.

- (1) Subset: security of set containment implies security of it
- (2) Empty: security of set disjointness implies security of it
- (3) Completeness: let $X'_i = X_i - I$; then, security of set disjointness of X'_i 's implies security of it
- (4) Intersection: let $X'_i = X_i - I$; then, security of set disjointness of X'_i 's and security of set containment of (I, X_1) and (I, X_2) , imply security of it
- (5) Membership: security of set containment implies security of it
- (6) Nonmembership: security of set disjointness of set $\{y\}$ and set X implies security of it

From all the above, we complete the proof of security of our scheme. $\square$

7. Performance Evaluation

In this section, we evaluate the performance of the PvCT framework for contact tracing processing. Three datasets are used in the experiments:

- (i) Foursquare and Gowalla (see [37]): the Foursquare dataset contains 194,108 data records and the Gowalla dataset contains 456,967 data records, which are the users' trajectory information. Each record has the form of $\langle [\text{longitude}, \text{latitude}], \text{check/in place}, \text{timestamp} \rangle$.
- (ii) Transportation: the transportation dataset is extracted from the public transport information website such as railway and bus during the period from Oct 1st, 2020, to Oct 15th, 2020, nationwide. It contains over 20,000 transportation records. Each transport record is in the form of $\langle \{\text{stations}\}, \text{vehicle}_{\text{ID}}, \text{timestamps} \rangle$, where $\{\text{stations}\}$ and timestamp are all the stations which the related vehicle will pass through and the corresponding time, $\text{vehicle}_{\text{ID}}$ is train number or bus plates.

The client and authority is set up on a commodity laptop computer with Intel Xeon E5-2603 CPU and 8 GB RAM, running on Ubuntu. The CSPs are set up on an Intel Xeon E5-2680 CPU, 2.8 GHz, and 256 GB RAM, running on Ubuntu 14.04.6. The experiments are written in C, and the following libraries are used: PBC for bilinear pairing computation, Crypto++ for 160 bit SHA-1 hash operations.

To evaluate the performance of verifiable computation in contact tracing, we mainly use four metrics:

- (i) Proof generation cost in terms of CSP CPU time
- (ii) Result verification cost in terms of client CPU time

- (iii) Size of the VO transmitted from the CSP to the user
- (iv) Storage overhead in the client side

The results are reported based on an average of 10 randomly generated operations.

7.1. Comparison with Other Work. We compare our algorithm against

- (1) Existing verifiable contact tracing algorithms
- (2) General verifiable computation algorithms

First, as introduced in Sections 1 and 2, to the best of our knowledge, there are two existing “verifiable contact tracing” algorithms [7, 8]. Nevertheless, the same term “verifiability” has different meanings between our work and theirs. In their paper, “verifiability” is with respect to access control. More specifically, what needs to be proved as well as what can be verified by the verifiers (some public authorities) in their works is the fact that a person indeed has access authorization to certain related contact tracing information. In other words, they focus on verifying whether clients have the authority to log in to the system as well as issue contact tracing queries. However, our paper focuses on verifying the accuracy and integrity of contact tracing query results. They actually deal with orthogonal issues, and our algorithms can also be utilized in their schemes to enhance security further. Therefore, due to the essential difference mentioned above, there is no need to give an experimental comparison between our verifiability property and theirs.

Secondly, to deal with this special verification problem, namely, verification of contact tracing result. There is a possible way to guarantee both accuracy and integrity, which is utilizing general verifiable computation algorithms such as Pinocchio [12] and Geppetto [38]. However, as we introduced in Section 1, general verifiable computation algorithms are not practical to apply in the contact tracing scenario. Here, we provide detailed analysis for different metrics between general verifiable computation algorithms and our algorithms as below:

Proof generation: in terms of proof generation procedure, those schemes are all based on the methodology that first translated the target function F to the corresponding arithmetic/Boolean circuit C , and they need to convert the arithmetic/Boolean circuit into a Quadratic Arithmetic/ Span Program (QAP/QSP). Based on these preprocessing, they conduct the following processing. So, the complexity of generating related proofs in these schemes is in proportion to the size of the circuit C as well as the length of the inputs. However, our schemes take advantage of the underlying algebraic structure, not only avoiding extra cost from introducing circuits but also having asymptotic complexity only in the length of the input. More details with respect to the differences between general circuit-based methods and special algebraic-based methods can be found here [39].

Moreover, it also takes $O(n \log(n))$ to convert a function F to a circuit C [40], which is also a substantial computation burden.

Verification: in terms of the verification procedure, a truth we would like to mention is that general verification computation schemes mainly focus on complicated function evaluation, such as large matrix multiplication. If utilizing these algorithms in the contact tracing scenario, they can only verify one record for each execution. Therefore, if there are m records in the result dataset, the complexity of verification procedure in general verifiable computation is $O(m)$. In our algorithm, the verification complexity for verifying a result dataset does not grow with m , and it only incurs another constant verification regardless of what number of records in the result dataset, which is $O(1)$. In this way, our algorithm is more efficient and compatible with the contact tracing scenario compared with the general VC schemes.

7.2. Performance of Set Accumulators. We first utilize two synthetic sets to evaluate the performance of the three set accumulators, *i.e.*, `subset(.,.)`, `empty(.,.)`, and `intersection(.,.)` in terms of

- (i) The proof generation time
- (ii) The verification time
- (iii) The proof size

We set the size of two sets to be 5,000 and select 20% to 50% of the set as target subset or intersection set. As reported in Table 3, the proof generation time is generally longer than verification time, but still acceptable due to this part is processed in the CSP side other than the client side. In contrast, the verification time and proof size are all constants, irrespective of the size of sets.

7.3. Verifiable Contact Tracing Performance. We evaluate the overall performance of publicly verifiable contact tracing algorithm on all three datasets. First of all, Table 4 shows the client’s setup cost for the ADS construction time and the ADS size. Although the setup time is a little higher than other algorithms of our ADS construction, it is only a one-time computation, being amortized during the following process.

To evaluate the performance for publicly verifiable contact tracing, we measure the two phases: Query Phase and Matching Phase separately. In the query phase evaluation, we vary the size of result set from 100 to 5,000, and all of them are randomly selected from Foursquare, Gowalla, and Transportation. The results for the query phase are shown in Figure 4. It is observed that CSP CPU time is generally linear to the size of result set, but still within 5.5 s. And, the client CPU time is always around 1.7 ms which is constant, and this suggests that our algorithm is robust against larger result set. Meanwhile, because the proof size is always constant, as shown in Table 3, the whole VO size that CSP returns to the client only depends on the size of the result set.

We next evaluate the performance for the matching phase. First, we assume the matching result is negative, *i.e.*, the client is not a close contact to the diagnosed people. That

TABLE 3: Performance of set accumulators.

Accumulator	Proof generation time (s)			Verification time (ms)	Proof size (bytes)
	20%	30%	50%		
Subset(...)	5.89	4.85	3.55	1.75	128
Empty(...)	7.24	11.45	16.52	2.65	256
Intersection(...)	45.67	31.54	19.39	6.19	512

TABLE 4: Setup overhead.

Dataset	Dataset size (MB)	ADS construction time (s)	ADS size(Bytes)
FourSquare	11.8	244	128
Gowalla	25.7	549	128
Transportation	46.1	1,235	128

FIGURE 4: Query phase performance.

also means the intersection between the result set and the set of diagnosed people is empty. We vary the size of result set from 100 to 3,700. As shown in Figure 5, the CSP CPU time is in proportion to the size of result set and within 30 s. It is interesting to note that the cost of CSP CPU time is relatively higher for the matching phase than the query phase. This is caused by the computation of extended Euclidean polynomial, which incurs more overhead when computing the proof on the CSP. And, the client CPU time is also constant and around 2.7 ms. Meanwhile, because there is no intersection between the two sets, the VO size is independent of the size of result set and only costs 256 bytes.

Then, we evaluate the performance when the matching result is positive, *i.e.*, the client is a close contact to the diagnosed people. And, this can contain two different situations as we described in Algorithm 2, the first situation is the intersection I^* between transportation datasets of the client, and the diagnosed people is not empty; then, the CSP CPU time contains both the proof generation time of I^* and I . The second situation is the intersection I^* is empty; then, the CSP CPU time only contains the proof generation time of I . Since there is no difference between these two proof generation processes, we do not evaluate them separately. We vary the size of the intersection from 200 to 6,000, and this size can be viewed as the size of I or the size of $I \cup I^*$. As shown in Figure 6, the CSP CPU time and the size of the intersection are negatively correlated. This is because the main

computation is to do the extended Euclidean polynomial over the supplementary set of the intersection. Therefore, the larger the intersection size is, the faster the computation will process. At the meantime, the client CPU time is also near constant and around 6.3 ms. The VO size depends on the size of the intersection.

There is an optimization scheme (MP*) that we can utilize in real-world applications. That is when the client only wants to know the individual he issued is a positive contact or not and does not want to know the accurate intersection between the two datasets. Then, instead of generating a proof for the whole intersection I and I^* , CSP only needs to generate a proof that can prove one random record of the intersection belongs to both the client's records set and the diagnosed people's records set. And, the client can easily check that whether the chosen record is belonging to his or not; then, the CSP only needs to prove that the chosen record also belongs to the diagnosed people. As shown in Figure 6, the CSP CPU time is highly reduced compared to the original version, and it only costs a constant time around 6.07 ms. And, the client CPU time is also reduced to around 1.73 ms caused by the client only needs to do the simple subset verification computation where the subset only contains 1 record. Then, we can easily find out the VO size is also almost constant as the VO only contains 1 record (about 1 KB) and its proof (128Bytes).

Finally, we evaluate the storage overhead in the client side. Instead of storing the full version of all the records in

FIGURE 5: Matching phase performance (negative matching result).

FIGURE 6: Matching phase performance (positive matching result).

the client side, the client only needs to store the ADS AccDigest in the client side, and it only costs 128Bytes; compared with the real-world datasets Foursquare (11.8 MB), Gowalla (25.7 MB), and Transportation (46.1 MB) which we utilized in our experiment, it highly reduces the heavy storage burden in the client side.

8. Conclusion

In this paper, we study the problem of publicly verifiable contact tracing in cloud computing. To achieve both accuracy and integrity of contact tracing results, we develop a novel set accumulator-based ADS scheme that enables efficient verification and low storage overhead. Based on this building block, we propose the PvCT framework and give a detailed discussion of different contact tracing phases. The robustness of the proposed building block is substantiated by rigorous security proof based on q -Strong Bilinear Diffie–Hellman assumption. Empirical results over three real-world datasets show that our algorithm is practically feasible within milliseconds of client CPU time and can reduce the storage overhead from the size of datasets to a constant.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The work was supported by the National Natural Science Foundation of China (no. 61972094, 61902299, 61976168, and 62032005), National Key R&D Program of China (no. 2018YFC0831200), project funded by China Postdoctoral Science Foundation (no. 2018M633473, 2019TQ0239, and 2019M663636), Key Research and Development Plan of Shaanxi Province (no. 2019ZDLGY13-09), S&T Program of Hebei (no. 20310102D), Natural Science Basic Research Program of Shaanxi Province (no. 2019CGXNG-023), and CCF-Huawei Database System Innovation Research Plan (no. CCF-HuaweiDBIR008B) and in part by the young talent promotion project of Fujian Science and Technology Association.

References

- [1] WHO, *Implementation and Management of Contact Tracing For Ebola Virus Disease*, WHO, Geneva, Switzerland, 2015, <https://www.who.int/csr/resources/publications/ebola/contact-tracing/>.
- [2] C. Criddle and L. Kelion, “Coronavirus contact tracing: world split between two types of app,” 2020.

- [3] R. Raskar, I. Schunemann, R. Barbar et al., "Apps gone rogue: maintaining personal privacy in an epidemic," 2020, <https://arxiv.org/abs/2003.08567>.
- [4] N. Trieu, K. Shehata, P. Saxena, R. Shokri, and D. Song, "Epione: lightweight contact tracing with strong privacy," 2020, <https://arxiv.org/abs/2004.13293>.
- [5] H. Cho, D. Ippolito, and Y. W. Yu, "Contact tracing mobile apps for covid-19: privacy considerations and related trade-offs," 2020, <https://arxiv.org/abs/2003.11511>.
- [6] J. Bay, J. Kek, A. Tan et al., "A privacy-preserving protocol for community-driven contact tracing across borders," Technical Report, Government Technology Agency, Singapore, 2020.
- [7] H. Xu, L. Zhang, O. Onireti, Y. Fang, W. J. Buchanan, and M. A. Imran, "Beepttrace: blockchain-enabled privacy-preserving contact tracing for covid-19 pandemic and beyond," *IEEE Internet of Things Journal*, vol. 8, no. 5, pp. 3915–3929, 2021.
- [8] A. Canidio, G. Costa, and L. Galletta, "Private-yet-verifiable contact tracing," 2020, <https://arxiv.org/pdf/2010.09303.pdf>.
- [9] M. D. Ryan, "Cloud computing privacy concerns on our doorstep," *Communications of the ACM*, vol. 54, no. 1, pp. 36–38, 2011.
- [10] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza, "Snarks for c: verifying program executions succinctly and in zero knowledge," in *Proceedings of the Annual Cryptology Conference*, pp. 90–108, Santa Barbara, CA, USA, August 2013.
- [11] T. Xie, J. Zhang, Y. Zhang, C. Papamanthou, and D. Song, "Libra: succinct zero-knowledge proofs with optimal prover computation," in *Proceedings of the Annual International Cryptology Conference*, pp. 733–764, Santa Barbara, CA, USA, August 2019.
- [12] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: nearly practical verifiable computation," in *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, pp. 238–252, Berkeley, CA, USA, May 2013.
- [13] R. Tamassia, "Authenticated data structures," in *Proceedings of the European Symposium on Algorithms*, pp. 2–5, Sophia Antipolis, France, September 2003.
- [14] A. Miller, M. Hicks, J. Katz, and E. Shi, "Authenticated data structures, generically," *ACM SIGPLAN Notices*, vol. 49, no. 1, pp. 411–423, 2014.
- [15] C. Xu, J. Xu, H. Hu, and M. H. Au, "When query authentication meets fine-grained access control: a zero-knowledge approach," in *Proceedings of the 2018 International Conference on Management of Data*, pp. 147–162, Goa, India, January 2018.
- [16] Q. Chen, H. Hu, and J. Xu, "Authenticated online data integration services," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pp. 167–181, Portland OR USA, June 2015.
- [17] H. Pang and K.-L. Tan, "Authenticating query results in edge computing," in *Proceedings. 20th International Conference on Data Engineering*, pp. 560–571, Boston, MA, USA, December 2004.
- [18] Apple, "Privacy preserving contact tracing," 2020, <https://www.apple.com/covid19/contacttracing2020>.
- [19] Google, "Exposure notification api," 2020, <https://www.google.com/covid19/exposurenotifications/2020>.
- [20] J. Chan, S. Gollakota, E. Horvitz et al., "Pact: privacy sensitive protocols and mechanisms for mobile contact tracing," 2020, <https://arxiv.org/abs/2004.03544>.
- [21] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin, "Dynamic authenticated index structures for outsourced databases," in *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, pp. 121–132, Chicago IL USA, June 2006.
- [22] Y. Yang, S. Papadopoulos, D. Papadias, and G. Kollios, "Authenticated indexing for outsourced spatial databases," *The VLDB Journal*, vol. 18, no. 3, pp. 631–648, 2009.
- [23] Y. Zhang, D. Genkin, J. Katz, D. Papadopoulos, and C. Papamanthou, "vsq: verifying arbitrary sql queries over dynamic outsourced databases," in *Proceedings of the 2017 IEEE Symposium on Security and Privacy (SP)*, pp. 863–880, San Jose, CA, USA, May 2017.
- [24] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Succinct non-interactive zero knowledge for a von neumann architecture," in *Proceedings of the 23rd {USENIX} Security Symposium {USENIX} Security*, pp. 781–796, Berkeley, CA, USA, 2014.
- [25] J. Groth, "On the size of pairing-based non-interactive arguments," in *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 305–326, Zagreb, Croatia, November 2016.
- [26] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Scalable zero knowledge via cycles of elliptic curves," *Algorithmica*, vol. 79, no. 4, pp. 1102–1160, 2017.
- [27] M. Maller, S. Bowe, M. Kohlweiss, and S. Meiklejohn, "Sonic: zero-knowledge snarks from linear-size universal and updatable structured reference strings," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pp. 2111–2128, London, UK, November 2019.
- [28] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Scalable zero knowledge with no trusted setup," in *Proceedings of the Annual International Cryptology Conference*, pp. 701–732, Santa Barbara, CA, USA, August 2019.
- [29] R. C. Merkle, "A certified digital signature," in *Proceedings of the Conference on the Theory and Application of Cryptology*, pp. 218–238, Auckland, New Zealand, December 1989.
- [30] S. Chatterjee and A. Menezes, "On cryptographic protocols employing asymmetric pairings - the role of Ψ revisited," *Discrete Applied Mathematics*, vol. 159, no. 13, pp. 1311–1322, 2011.
- [31] D. Boneh and X. Boyen, "Short signatures without random oracles," in *International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 56–73, Interlaken, Switzerland, May 2004.
- [32] C. Papamanthou, R. Tamassia, and N. Triandopoulos, "Optimal verification of operations on dynamic sets," in *Proceedings of the Annual Cryptology Conference*, pp. 91–110, Santa Barbara, CA, USA, August 2011.
- [33] F. P. Preparata and D. V. Sarwate, "Computational complexity of fourier transforms over finite fields," *Mathematics of Computation*, vol. 31, no. 139, p. 740, 1977.
- [34] Y. Zhang, J. Katz, and C. Papamanthou, "An expressive (zero-knowledge) set accumulator," in *Proceedings of the 2017 IEEE European Symposium on Security and Privacy (EuroSecP)*, pp. 158–173, Paris, France, April 2017.
- [35] R. Canetti, O. Paneth, D. Papadopoulos, and N. Triandopoulos, "Verifiable set operations over outsourced databases," *Public-Key Cryptography - PKC 2014*, vol. 83, pp. 113–130, 2014.
- [36] Y. Peng, L. Wang, J. Cui et al., "A lightweight and forward-secure range query on geographically encrypted data," *IEEE Transactions on Dependable and Secure Computing*, vol. 17, 2020.

- [37] Q. Yuan, G. Cong, Z. Ma, A. Sun, and N. M. Thalmann, "Time-aware point-of-interest recommendation," in *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 363–372, Dublin, Ireland, July 2013.
- [38] C. Costello, C. Fournet, J. Howell et al., "Geppetto: versatile verifiable computation," in *Proceedings of the 2015 IEEE Symposium on Security and Privacy*, pp. 253–270, San Jose, CA, USA, May 2015.
- [39] M. Chase, C. Ganesh, and P. Mohassel, "Efficient zero-knowledge proof of algebraic and non-algebraic statements with applications to privacy preserving credentials," in *Proceedings of the Annual International Cryptology Conference*, pp. 499–530, Santa Barbara, CA, USA, August 2016.
- [40] N. Pippenger and M. J. Fischer, "Relations among complexity measures," *Journal of the ACM*, vol. 26, no. 2, pp. 361–381, 1979.