

# A Post-Quantum Four-Party Outsourced Authentication

Reza Ghasemi  
Bu-Ali Sina University  
Hamedan, Iran  
Email: r.ghasemi@basu.ac.ir

Alptekin K p u  
Ko  University  
İstanbul, Turkey  
Email: akupcu@ku.edu.tr

**Abstract**—In this paper, for the first time, we consider a four-party scenario, where a *customer* holding a smart card wants to authenticate herself to a *server*, which employs a *cloud database* to verify the customer. The customers initially register with the *manager*. The manager outsources the processed registration data to a cloud database. Then, the customer only interacts with the server for authentication purposes. The server employs the help of the cloud database during authentication. In addition to the security of the authentication, privacy is another goal, where the cloud should not learn which customer is interacting with which server. We consider several different threat models and provide secure and efficient generic solutions as well as a post-quantum secure solution.

## I. INTRODUCTION

**Motivation:** In the standard two-party authentication scenario, there is a customer and a server that execute all parts of registration and authentication. In this scenario, a customer first registers with the server, then, at a later time, (s)he can authenticate with the server through an authentication protocol. This scenario should be repeated for any server that a user wants to authenticate themselves. This incurs a high demand for a local storage to store authentication parameters. Take typical password-based authentication schemes as an example. In these schemes, we need to store a password with regard to any server, ends up with a long list of passwords whose management would be hard. Not only do servers need a large local storage to keep registered passwords, but also customers should memorize their passwords with different servers. Hence, management issue happens in both sides.

Quantum algorithms provide attackers with new means so that some cryptosystems cannot resist against quantum attacks. Therefore, not only should we be cautious of using current authentication protocols, but also we should develop new quantum resistant authentication protocols.

A naive solution to mentioned problems is a centralized database that stores authentication parameters which are used in the authentication stage. This can address management problem but privacy concern arises. The privacy of users in typical authentication protocols usually is violated since in authentication process, it will be understood who is trying to authenticate him/herself.

Above summary inclined us to propose a privacy-preserving centralized password manager that can handle users authentication parameters and it is quantum resistance ensuring us

that the advent of quantum computers does not jeopardize the stored password’s security.

**Scenario:** Consider a high security bank with multiple branches. Customers can authenticate with any branch to reach their accounts or bank safes. Essentially, we are considering the offline case, where the customer is physically present, trying to access the bank account or safe deposit box, and hence the performance requirement is not real-time as in the online case. The authentication procedure may be performed while the customer is waiting for the teller to announce their turn. To enable such access, the trivial solution would be to employ two-party solutions. In that case, each branch needs to keep a copy of the customer’s data and play the role of the server in the authentication protocol. Unfortunately, this solution has several shortcomings. Firstly, this imposes high maintenance and security costs to a bank with many branches. Secondly, this increases the attack surface. Thirdly, if a branch updates a record, consistency issues should be handled. Moreover, consider the case that the customers are not just authenticating against several branches of a bank, but against multiple rival banks using a government-regulated customer database. Then, further privacy issues arise, such as hiding “when which customer visits which bank”.

**Three- and Four-Party Authentication:** In this paper, we consider three- and four-party scenarios. As in the two-party case, the *customer* wants to authenticate with the *server*. But, there is a third party, a *cloud* database, employed by the server to facilitate authentication. In the bank branches scenario above, the bank may create a cloud database such that each branch has access to it. This cloud database enables each branch to act as the server in the authentication protocol with help from the cloud. Moreover, as in the multiple banks scenario above, it is possible that there is a fourth party, the *manager*, who carries the duty of registering customers. In the example above, the manager would be the government who regulates the database. The three-party scenario was previously partially studied (see below), whereas the four-party scenario is introduced anew in this paper.

**Privacy:** Considering outsourcing authentication, the three-party scenario was considered before, where previous works proposed using a centralized database [1, 2, 3, 4]. The identification and authentication processes are outsourced to a centralized entity enjoying high computational power and

storage. Generally in these schemes, the extracted biometric or genomic traits are passed to the centralized database. The main issue related to such methods is that the database may infer some information about the customers. Hence, this scenario is only applicable when the database is fully trusted. Even when the customer data is kept confidential, each time the customer wants to authenticate, the database would be contacted, and this may breach the privacy of the user, since the database would learn *when which* customer uses *which* service(s). Consider, for example, a high-ranking military officer obtaining access. By simply obtaining the database logs from the cloud, the enemy may infer the behavioral patterns of the high-ranking officer and plan out their attack accordingly.

In our model, the database can be operated by a cloud provider that offers databases with large storage and high computational power at reduced costs, as well as high availability and reliability. These companies are not thoroughly trusted. Therefore, they cannot take on the role of the database in the previous three-party authentication or identification solutions. In this paper, we consider several threat models within our three- and four-party model, where the cloud database is not fully trusted, and provide both general and post-quantum secure solutions protecting the privacy of the customers and servers, as well as the security of the authentication.

#### A. Detailed Problem Statement

Our objective is to create a protocol (or rather, a framework of protocols), where the authentication parameters of the customers are accumulated securely in a cloud database in such way that these confidential data are used for authentication purposes. In other words, the customers, whose registration information have been stored in the cloud database, can prove to the servers that they belong to this dataset. The main structure and the entities involved are illustrated in Figure 1.

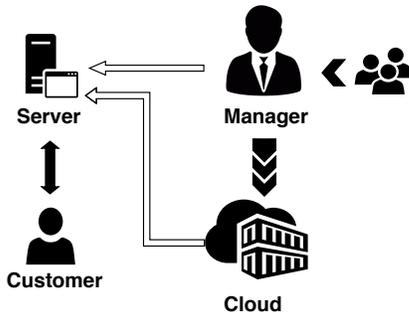


Fig. 1: Proposed four-party structure consisting of four entities *Manager*, *Customer(s)*, *Server(s)* and *Cloud*.

In our **four-party model**, we have four different entities: *Cloud Database*, *Servers*, *Customers* and a trusted *Manager*.

- **Cloud:** All data used in the authentication phase are stored in a (shared) database during registration. The stored data are confidential, and even the database itself should not have direct access to the underlying data. Moreover, customers' and servers' privacy against the

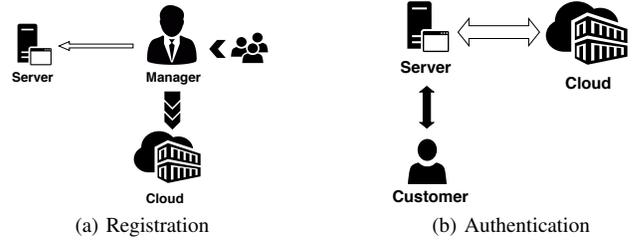


Fig. 2: Entities' communication in Authentication and Registration stages.

cloud should be protected so that the cloud does not learn which customer gets service from which server when.

- **Server(s):** These are the entities that the customers want to authenticate with so that they can get service. In a regular two-party authentication scenario, the database would have been under the full control and trust of the server. In our four-party model, the database is on the cloud, and the server is a separate entity. During authentication, the servers are allowed to contact the cloud database, but we assume they do not have a full copy of the database at hand, due to various reasons underlined in the introduction (e.g., cost, central management, government intervention).
- **Customer(s):** They are the end users who want to employ the servers' services. Therefore, they need to prove their access rights to the servers. In our post-quantum secure construction, we assume that the customers carry a smart card to use during the authentication process. Therefore, they can carry out some computation as well.
- **Manager:** As a trusted entity, the manager performs the registration of the customers, and delivers the corresponding registration data to the cloud. Furthermore, the manager provides the servers with the latest digest of the stored records in the cloud. This is for preventing the untrusted cloud from adding unauthorized records or removing or modifying existing ones. The manager needs to be trusted, since also in the existing (two-party) authentication solutions, the registration phase must be trusted [5]. Otherwise, any malicious user may already be registered to the database, and hence discussing security in the setting where the manager is adversarial becomes obsolete.

Our model is comprised of two main stages: i) **Registration**, and ii) **Authentication** (Figure 2). In the *registration* stage, the customers register their information by interacting with the manager. After processing the data, the database will be outsourced to the cloud, and the servers will be informed of the digest by the manager. As in the previous works, it is assumed that the registration is performed honestly [5, 6], since if a malicious user manages to register, then there is no way to prevent their login. The second stage is the *authentication* process, where a customer tries to convince a server of its authenticity. As illustrated, the manager is *not* involved, and the customer *only* contacts the server, who gets help from

the cloud. This is because of efficiency and ease of use. Eventually, the server is able to validate the claim of the underlying customer (whether or not the customer belongs to the cloud database).

Our four-party model also encompasses a **three-party model**, where there is no manager, and hence the registration goes through the server (or the cloud) instead. There is still the cloud database for outsourcing purposes, and we still have the same security and privacy requirements regarding the cloud database. Similar three-party settings were previously considered by several related works in the password-based setting [5, 7, 6, 8].

**Goals:** For security and privacy, the solution should provide the following protection (for game-based formalization of these properties, see Section VI): `nolistsep, noitemsep`

- **Completeness:** Authorized customers (those registered to the database) should be able to convince an honest server.
- **Soundness:** Unauthorized customers (those who are not registered to the database) should not be able to convince an honest server.
- **Confidentiality:** Confidential customer information must be securely outsourced to the cloud database such that even the cloud cannot obtain the underlying data.
- **Hack-Resistance:** If the database is stolen and the adversary has a copy of the stored data, the attacker should still not be able to impersonate a customer. Note that confidentiality property above protects the customer data against hackers, but it does not necessarily imply hack-resistance.
- **Non-impersonation:** The *servers* must not be able to impersonate a customer against other servers after interacting with her. Note that in the standard two-party setting, usually the servers are trusted. For example, in the traditional password-based authentication, the servers obtain a copy of the customer’s password. When the same password is used elsewhere, such impersonation is possible [9]. In our new model, we require this property to protect the customer further.
- **Privacy:** The *cloud* should not be able to figure out which customer tries to authenticate. Observe that since we consider the authentication scenario, the server needs to know which customer is authenticating. But, for the sake of privacy of the customer, we would like to hide this information from the cloud, as it does not “need to know”. Privacy usually includes **Unlinkability**, which says that the cloud should not be able to distinguish multiple authentications by the same customer from that of different customers.
- **Non-replication:** The servers should ideally not be able to obtain a full copy of the database after interacting with multiple customers (and the cloud). Ideally, the only thing that the servers will understand is whether a customer is an authorized person or not.

We point that not all of these properties may be necessary in every scenario. Yet, we keep both our definitions

and framework as general and secure as possible so that it can address as many scenarios as possible. Later on, when we discuss our constructions’ details, we emphasize which construction satisfies which security and privacy goals, and sometimes even suggest modifications to our protocols such that they can provide more (or fewer) goals with less (or better) efficiency.

**Adversarial Model:** We consider different threat models where parties can be semi-honest (they exactly follow the protocol but try to infer additional information beyond what has been defined in the protocol) or some of them can be fully malicious. Since we consider an authentication scenario, we assume that the customer is malicious and the server is semi-honest during authentication (as otherwise a malicious server can simply accept any authentication attempt). Moreover, as the manager is only involved during the registration and the registration needs to be trusted, the manager is assumed to be honest. The cloud is semi-honest in the authentication stage, but it collaborates with the malicious customers in the sense that malicious customers can read the whole stored data and possibly try adding unauthorized customers’ information to the database. We leave handling a fully malicious cloud as future work.

While we allow the adversary to also control the communication channels, we do require that the server and the cloud obtain the authentic values from the manager during registration (via an authenticated channel or using digital signatures). Moreover, when a customer contacts a server, we assume a secure and server-authenticated channel (e.g., TLS) is employed.

## B. Contributions

In this paper, `noitemsep, nolistsep`

- we define three- and four-party authentication models, where the three-party model was previously used by [5, 10, 11, 2, 12, 7, 13, 14, 3, 6, 15, 8, 16, 1, 17] with limitations or under different settings, and the four-party model is novel.
- we show how to convert any two-party authentication solution to work under the three- or four-party model generically with the help of oblivious transfer, preserving the underlying security properties, and achieving even further security goals such as privacy.
- we provide a lattice-based post-quantum secure specific construction that achieves all our security goals against a malicious customer, semi-honest server, and a semi-honest cloud. In the proposed lattice based construction, there is no need to use oblivious transfer to meet the objectives. We also experimentally evaluate the performance of this solution.
- we provide game-based security definitions and reduction proofs for our security goals (soundness, confidentiality, hack-resistance, non-impersonation, and privacy), and briefly discuss completeness and non-replication goals that are obviously achieved by our construction.

## II. RELATED WORKS

**Password-based Authentication and Key Exchange:** Passwords are the most commonly employed form of authentication. Unfortunately, in the two-party setting, they are vulnerable to many attacks, including offline dictionary attacks, man-in-the-middle attacks, phishing, and honeypots [5]. In the three-party setting, where the customer employs the help of a cloud or a mobile device, solutions providing provable security were proposed first by [5], and then improved to the multi-party setting by [6]. In the related scenario, passwords may be employed to create a shared key between the client and the server [18, 19, 20]. This password-authenticated key exchange setting was also analyzed in multi-party settings [7, 8].

**Outsourced Authentication:** Many existing schemes just employ a trusted outsourcing scenario, where the query about a customer is sent to the cloud, and the cloud decides on the authentication result [12, 16, 10, 15, 14, 11]. These schemes do not provide security or privacy guarantees against the cloud, since it is assumed to be trusted, which is in contrast to our case.

More privacy-aware solutions were also proposed. Some solutions [2, 13, 17] employ fully- or somewhat-homomorphic encryption schemes or secure multi-party computation techniques [1]. Thus, they achieve good security but with large cost. Another interesting idea is to employ tokens extracted from the users' traits [3]. It is possible that the tokens are computed by the servers or the customers. But, even when the customers generate the tokens, they are deterministic, and hence the cloud would learn which user tries to authenticate when, and therefore privacy cannot be provided.

## III. PRELIMINARIES

**Notation:**  $0^l$  is used to depict a sequence of  $l$  concatenated zeros and  $\parallel$  denotes concatenation. Notice that for the sake of simplicity, all vectors in this paper are vertical and transpose notation is omitted.

**Accumulators:** An accumulator contains  $k$  items summarized in a digest, and the holder of the digest can ask (non-)membership queries [21, 22]. In return, a (constant size) witness would be provided, and hence the answer of the query is verifiable. Dynamic accumulators enable adding or removing new items, while updating the digest and the witnesses of the other items accordingly [23, 24].

For our purposes, an  $AccuGen(1^\lambda, k) \rightarrow (apk, ask)$  algorithm generates the public-secret key-pair of the accumulator with capacity  $k$ , where  $\lambda$  is a security parameter provided in unary as input. Then,  $AccuAdd(apk, \mu, itemtoadd) \rightarrow \mu'$  and  $AccuDelete(apk, \mu, itemtoremove) \rightarrow \mu'$  functions take the public key and the current accumulator with its digest  $\mu$ , add or delete some item, and output the new digest, as well as the updated witnesses for all the items (for our purposes, the witnesses are part of the internal state and we do not explicitly show in the  $AccuAdd$  and  $AccuDelete$  notation). Note that the witnesses may be updated without the secret key. Finally, an  $AccuVerify(apk, \mu, item, wit) \rightarrow 0/1$  algorithm takes the public key and the digest of the accumulator, the item

that is queried, and the witness  $wit$ , and outputs rejection or acceptance.

**Oblivious Transfer (OT):** An oblivious transfer is a two-party protocol, where there is a sender and a receiver [25, 26]. The sender holds  $s$  items, and the receiver usually has an index  $1 \leq i \leq s$  such that once the protocol is over, the sender learns nothing about  $i$ , whereas the receiver learns item  $i$  and nothing else. In particular, multiple executions of an OT protocol are unlinkable; even when the same index  $i$  is employed multiple times, the sender cannot detect this. Some OT protocols allow  $i$  values to be any bit string (rather than just consecutive integers) [27]. The important property is that the  $i$  values are unique.

**Secret Sharing:** Secret sharing is a method to distribute a confidential value among several participants by assigning each party a share such that a predetermined authorized subset of the participants are able to recover the original value using their shares [28, 29]. One advantage of these schemes is that they decrease data loss probability (when used with proper parameters). In addition, the allotted shares do not leak any information about the shared value, which prevents the engaged parties from gaining meaningful information about the shared value unless they constitute an authorized set of participants. For our purposes, we will denote  $t$ -out-of- $n$  threshold secret sharing, where any  $t$ -size subset of the  $n$  participants can reconstruct the original secret, with the notation  $Share(secret, rand, t, n) \rightarrow (share_1, \dots, share_n)$ . A simple 2-out-of-2 secret sharing may be obtained easily as  $share_1 = rand$  and  $share_2 = secret \oplus share_1$ .

**Lattice Based Identification:** Lattice based cryptography was introduced by Ajtai [30] in his seminal work. Protocols that are based on hard Lattice problems resist quantum attacks due to the fact that there is no known polynomial-time quantum algorithm to break them, whereas factoring and discrete logarithm problems are easy to break using quantum computers [31].

Roughly speaking, a lattice of dimension  $n$  is an additive subgroup of  $\mathbb{R}^n$ . An important hard problem in a lattice is the Shortest Independent Vectors Problem (SIVP), which is currently intractable even by quantum computers. This problem is defined as follows:

**Definition 1.** [32] *Given a lattice  $\mathcal{L}$  of dimension  $n$ , find  $n$  shortest linearly independent vectors that constitute a basis for the lattice.*

Another problem, based on which we construct our schemes, is the Small Integer Solution (SIS) problem below.

**Definition 2.** [33] *SIS(A): Given a matrix  $A \in \mathbb{Z}_p^{n \times m}$ , find two different vectors  $z \neq z' \in \mathbb{Z}^m$  such that  $Az = Az' \pmod p$  and  $\|z\|, \|z'\| \leq 10m^{1.5}$ .*

Although the SIS problem is not a lattice problem, there is a connection between the hardness of solving the SIS problem and that of the SIVP problem:

**Theorem 1.** [33] *For integer  $m = \lceil 4n \log n \rceil$  and some integer*

$p = \tilde{O}(n^3)$ , if there is a polynomial algorithm that can solve the SIS problem for a given uniformly random matrix  $A \in \mathbb{Z}_p^{n \times m}$ , then the SIVP problem can be solved within a polynomial factor of  $\tilde{O}(n^2)$ .

In other words, if appropriate parameters are chosen, then solving the SIS problem is infeasible as long as there is no polynomial solution for the SIVP problem.

Lyubashevsky proposed an identification protocol that is secure under active attacks based on the SIS problem [33]. In his scheme, a party that holds a binary vector  $\tilde{\omega}$  can prove that (s)he has access to  $\tilde{\omega}$  without leaking information about  $\tilde{\omega}$ , while the verifier just knows some matrix  $A$  and the vector  $\omega = A\tilde{\omega}$ . Since we employ this protocol in our solutions, we present Lyubashevsky's scheme briefly in Appendix A.

#### IV. GENERIC OUTSOURCED AUTHENTICATION SOLUTIONS

In this section, we show how to generically convert any two-party authentication scheme to a three- or four-party authentication scheme that meets our security objectives.

##### A. Three-Party Generic Construction

Remember that in the standard two-party authentication setting, there is a customer and a server. During the registration phase, the server learns some registration information about the customer. During the authentication phase, the customer proves that (s)he is a registered customer, which is verified against the registration information previously recorded.

In our model, we want to provide three- and four-party authentication. The third party is the cloud database, who is now supposed to keep the registration information on behalf of the server. The fourth party is the manager, who is trusted to register the customers. When there is no manager, the registration should go through the server. For simplicity, we first consider the three-party scenario without a trusted manager. This may be useful, for example, when the database is not shared, but the cloud is used for outsourcing purposes (e.g., for computational efficiency or decreasing storage costs).

In the case where the servers and the cloud are considered semi-honest, one may transform any secure two-party authentication solution to the three-party case as follows:

- 1) The customer and the server run the registration protocol of the two-party authentication technique. Assume that the customer has an identifier number  $i$  (which can be thought of as the unique username) and the server obtains the registration information  $info_i$ . The server then outsources the registration information  $info_i$  to the cloud.<sup>1</sup>
- 2) When customer  $i$  wants to authenticate with the server, the customer sends  $i$  to the server.
- 3) The server and the cloud run an oblivious transfer protocol, where the server's input is  $i$  and output is the registration information  $info_i$  regarding the customer  $i$ .

<sup>1</sup>It is also possible that the customer registers with the cloud directly, depending on the use case.

- 4) Now that the server knows the registration information  $info_i$  of the customer  $i$ , the customer and the server run the regular two-party authentication phase.

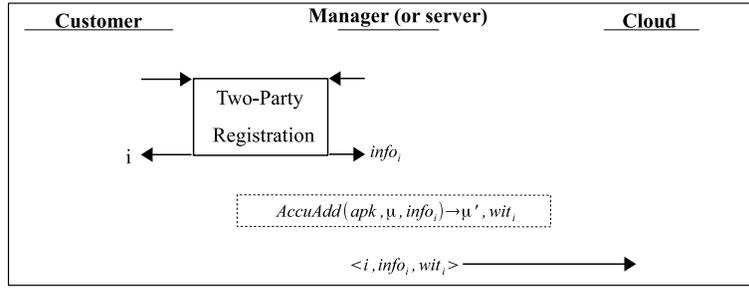
Security properties immediately follow from the security of the underlying two-party authentication protocol. Thus, this generic construction would have the same level of security as the underlying two-party solution. Specifically, *completeness* and *soundness* follows from the completeness and soundness of the two-party solution. If the underlying two-party protocol provides *confidentiality*, *hack-resistance*, and *non-impersonation*, so does our three-party protocol. Finally, *privacy* against the cloud during authentication immediately follows from the oblivious transfer protocol. Therefore, we do not provide a separate proof for this generic construction.

However, this scheme does not satisfy all of our aims. The first point that needs to be considered is that in this solution, the servers can obtain a copy of the database (thus non-replication is not achieved). Moreover, in the case that a malicious customer is able to insert fake records into the database, the server would be easily fooled into accepting unauthorized users. Thus, this solution only achieves soundness when the customer and the cloud does not collude. These reasons give us an incentive to create an advanced scheme that mitigates the presented concerns.

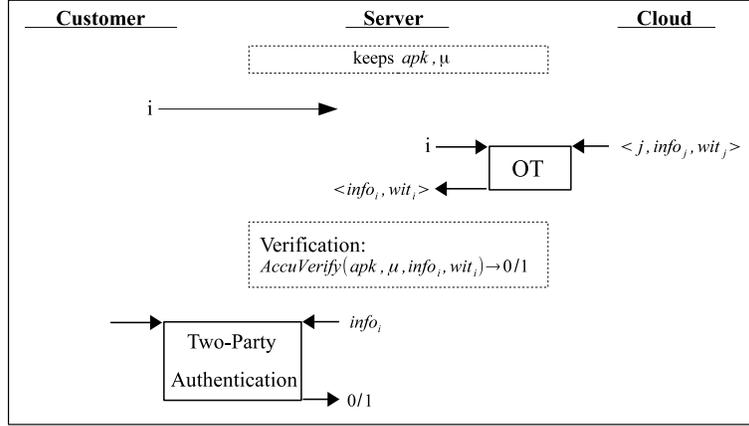
##### B. Four-Party Generic Construction

We assumed both the cloud and the server are semi-honest (while the customer is malicious) in the scenario above. Yet, since we consider an outsourcing scenario, it is possible that the cloud and the customer collude, and unauthorized customer records may be added to the database. In that case, we need to extend our generic construction as follows (see also Figure 3).

- 1) The customer registers through the manager in the four-party setting. The manager updates the cloud database with the registration information  $info_i$  of customer  $i$ . We also keep an accumulator over the customer registration database, and the accumulator digest  $\mu$  is updated accordingly using the *AccuAdd* algorithm with each new registration. The manager provides the server(s) with the latest digest  $\mu$  and the accumulator public key  $apk$ .
- 2) As in the three-party construction, when customer  $i$  wants to authenticate with the server, the customer first sends  $i$  to the server.
- 3) The server and the cloud run an oblivious transfer protocol as before. But in the extended version, for the value  $i$ , the server learns not only the registration information  $info_i$  regarding the customer  $i$ , but also the witness  $wit_i$  associated with that information. Upon receiving, the server immediately runs *AccuVerify*( $apk, \mu, info_i, wit_i$ ) and continues to the next step only when the verification succeeds. This ensures that the registration information at the database was not tampered with, and belongs to a legitimately registered user.



a) Registration.



b) Authentication.

Fig. 3: Authentication and Registration stages of our Generic Construction. The registration goes through the manager in the four-party setting, and the server in the three-party setting.

- 4) The customer and the server run the two-party authentication phase.

At a high level, we require the cloud to keep an accumulator over the registration information of each customer, where the digest is known by the server(s). The reason we pick the accumulator is that each witness in an accumulator is independent of each other, and hence can be provided by the sender as an additional value during the oblivious transfer. Thus, the cloud database provides  $(info_i, wit_i)$  pairs as input to the oblivious transfer protocol, and the  $i^{th}$  such pair would be learned by the server (who acts as the receiver in the oblivious transfer). Figure 3 visualizes the details of the registration and authentication phases. Finally, note that a customer may de-register (e.g., account deletion), in which case the manager runs the *AccuDelete* function updating the cloud database, and the server obtains the updated digest.

**Discussion:** Notice that, regarding the requirements that we described in Section I-A, there are potential improvements that one can obtain employing a tailored construction. The servers (or the cloud database) must not be able to impersonate a customer against other servers (*non-impersonation*). Moreover, the proposed protocol must protect stored data, even when the

database is stolen (*confidentiality* and *hack-resistance*). Again, these should be provided by the underlying protocol (e.g., [5]).

When the government sets up the shared database as the trusted manager, the servers obtain the latest digest from the government. Verifying the customer registration information against the trusted digest ensures the provided information by the cloud is authenticated. Together with the security of the underlying two-party authentication protocol, this enables *completeness* and *soundness* of our solution, even when a malicious customer colludes with the semi-honest cloud and tries to add unauthorized customer information to the database. Oblivious transfer provides the *privacy* goal, including *unlinkability*.

Another goal is *non-replication*: The servers should not be able to obtain a full copy of the database. This can be achieved via several methods. One technique is to use keyword-based oblivious transfer [27] and thus employ random customer identifier numbers  $i$  (and limit servers' attempts in trying to guess the customer identifier numbers). For this to work, it is assumed that the servers do not learn the customer identifiers during registration (i.e., the registration either is performed directly between the customer and the cloud database, or

through the trusted manager such as the government). Thus, depending on the underlying two-party authentication protocol employed, our generic four-party construction can achieve all the security and privacy properties we aimed at.

A better method for non-replication that we employ in our lattice-based construction is to retrieve a randomized response from the cloud, such that different randomness is used for each authentication attempt to prevent attacks. This way, one still needs access to the cloud database in the four-party scenario, and the cloud database is protected against replication by the servers. The details are provided in the following section.

**Efficiency:** It is possible to improve the efficiency of both of our generic constructions if the underlying application scenario does not require privacy. Essentially, the schemes use oblivious transfer for obtaining the **privacy** goal (including *unlinkability*) to prevent the cloud from identifying which user tries to authenticate herself/himself to a server. Therefore, in the use case if the cloud is trusted or leaking which customer performs the authentication is not important, the oblivious transfer part can be omitted and the server can directly ask the cloud for the stored data corresponding to customer  $i$ . This will result in a significant increase in the efficiency of the scheme, both at the server and the cloud.

## V. LATTICE-BASED CONSTRUCTION

We present a lattice-based post-quantum secure construction to prevent the servers from obtaining a copy of the whole database, achieving *non-replication* in addition to all our other security and privacy goals. This is done by ensuring that the cloud sends randomized responses that prevent duplication of the items in the database, and those responses cannot be re-used securely for authentication purposes. Therefore, servers cannot obtain the stored records and at the same time the cloud will not identify which user is going through the authentication process. In contrast to our generic four-party construction, in this scheme, oblivious transfer is not required, improving efficiency. In addition, the accumulator concept is leveraged to make sure that an attacking cloud fails to add or modify records in the database and play the role of a customer to convince the server. These two aims seem contradictory because the cloud provides the servers with randomized data used in the authentication process, and, at the same time, it should ensure the server of the soundness of the provided data. We achieve these objectives by utilizing novel lattice concepts.

**a) Setup.:** We require a one-time trusted setup (i.e., run by the manager).

$Setup(1^\lambda)$ : gets security parameter  $1^\lambda$  and chooses a public random matrix  $A$  of order  $n \times m$ , where  $m = \lceil 4n \log n \rceil$  and  $n = \lambda$ . Furthermore, a collision-resistant hash function  $H : \{0, 1\}^l \times \{0, 1\}^l \rightarrow \{0, 1\}^l$  is selected where  $l$  is equal to  $m/2$ . In addition, it runs  $AccuGen(1^\lambda, k)$  to obtain  $(apk, ask)$ , where  $k$  is an upper bound on the number of customers. The manager publishes  $A, n, m, l, \lambda, H, apk$  publicly.

**b) Registration.:** Figure 4 summarizes the registration stage.

$Register(1^\lambda, G_i)$ : Suppose that the authentication information regarding the  $i^{th}$  customer is  $G_i$ . Assume that  $Share$  is an algorithm which takes  $G_i \in \{0, 1\}^l$  and creates a 2-out-of-2 sharing of it outputting two binary vectors  $G'_i, G''_i \in \{0, 1\}^l$ . Consequently, neither  $G'_i$  nor  $G''_i$  individually leaks information about  $G_i$ . A simple algorithm for  $Share$  is,

$$Share(G_i, K_i, 2, 2) \rightarrow (G'_i, G''_i) = (G_i \oplus K_i, K_i)$$

where  $K_i$  is a random binary vector and will be stored in the customer's smart card. The value  $wit_i$  is a witness generated through the  $AccuAdd(apk, \mu, A[0^l || G''_i]) \rightarrow \mu'$  algorithm of an accumulator scheme during the registration phase by the manager. Then, the value  $SG_i = (G'_i, A[0^l || G''_i], wit_i)$  is passed to the cloud for storing in its database. The updated digest  $\mu'$  is sent to the servers by the manager. Notice that using this witness, the cloud is able to prove that  $A[0^l || G''_i]$  belongs to the database.

The value of  $K_i$  is kept inside the smart card of the  $i^{th}$  customer so that each time the customer provides a sample  $G_i$  for authentication, the same secret sharing can be re-performed using  $K_i$ . The smart card also holds the system parameters output by the  $Setup$ , namely  $A, n, m, l, \lambda, H$ , and is capable of performing some computation on these values.

**c) Authentication.:** In this phase, a customer attempts to prove that (s)he is a valid (registered) user. First, the server retrieves some randomized information from the cloud regarding the customer claiming (s)he is an authenticated entity. Second, an interactive authentication protocol between the server and the customer is executed to validate the claim. Figure 5 summarizes the authentication stage.

For clarity of the presentation, assume that the  $i^{th}$  customer tries to interact with the server. At this stage, the server retrieves information required for authenticating the  $i^{th}$  customer. But, one of our objectives is to prevent the server from replicating the database. Therefore, the following steps are executed by the three involved entities (see Figure 5 for details):

- 1) The customer sends  $i$  to the server.
- 2) All entities jointly choose at random a binary vector  $R_d$  [34, 35, 36].
- 3) For each record  $(G'_j, A[0^l || G''_j], wit_j)$ , the cloud computes  $A[H(G'_j, R_d) || 0^l]$ .
- 4) For each record, using Lyubashevsky's protocol [33], the cloud computes a non-interactive proof of knowledge  $proof_j$  that it has access to the  $H(G'_j, R_d) || 0^l$ .
- 5) Cloud sends all modified records  $A[H(G'_j, R_d) || 0^l], A[0^l || G''_j], wit_j, proof_j$  to the server.
- 6) The server verifies that  $wit_i$  is a valid witness for  $A[0^l || G''_i]$  (verified via  $AccuVerify(apk, \mu, A[0^l || G''_i], wit_i)$  using the latest digest received from the manager). If valid, the server also verifies the proof  $proof_i$  that the cloud knows  $H(G'_i, R_d) || 0^l$  corresponding to  $A[H(G'_i, R_d) || 0^l]$ .
- 7) The customer proves its identity by demonstrating that (s)he has  $H(G'_i, R_d) || G''_i$  through the Lyubashevsky's

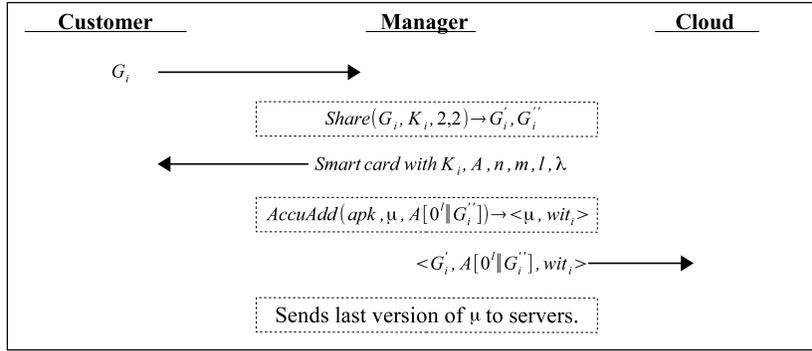


Fig. 4: Registration stage of our Lattice-based Construction.

authentication protocol (Table III in Appendix A). Notice that the server has  $A[H(G'_i, R_d) || G''_i]$  since

$$A[H(G'_i, R_d) || G''_i] = A[H(G'_i, R_d) || 0^l] + A[0^l || G''_i].$$

**Remark:** Because Lyubashevsky’s authentication protocol is a sigma protocol [37], it can be converted to a non-interactive zero knowledge proof of knowledge protocol using the Fiat Shamir heuristic [38] in the random oracle or common reference string models. The benefit of leveraging non-interactive proofs in our scheme is two-fold. First, since the cloud should not learn  $i$ , the cloud must add a non-interactive proof that it has access to  $H(G'_i, R_d) || 0^l$  to each record. The proof is then learned by the server without revealing  $i$ . Second, in addition, the customer is able to prove its authenticity through a non-interactive protocol. Consequently, using this technique, we are able to decrease the required number of transmission rounds, as seen in Figure 5, where the customer sends the non-interactive proof together with its identifier  $i$ . Therefore, as far as the customer is concerned, this results in a non-interactive authentication protocol (similar to simply sending the username and password). Thus, in the offline bank verification scenario, the customer provides this information to the teller via the smart card, and waits for the teller to announce her turn. The authentication protocol runs in the background while the customer is waiting. Finally, while it may be possible to employ other lattice-based proofs, our solution using the Lyubashevsky’s protocol is not black box: we modify the solution to fit our needs employing secret sharing over  $G_i$ .

**Losing the smart card:** The role of the smart card is storing randomly generated numbers used in sharing phase within registration process. These numbers carry no information about customers’ sensitive data. Consequently, losing smart card does not endanger the security or privacy of the customers. However, without smart cards, the customers cannot prove their identity to the servers and have to go through registration process and receive a new and valid smart card.

**Modular Architecture:** The proposed scheme is composed of different parts that can be adjusted to increase efficiency based on the requirements of the application that is considered.

Essentially, if privacy against the cloud is not a concern in the use case, instead of preparing and sending proofs for all the customers, the server can simply tell the value  $i$  to the cloud, and the cloud can simply send the information  $A[H(G'_i, R_d) || 0^l]$ ,  $A[0^l || G''_i]$  (together with the witness  $wit_i$  and the proof  $proof_i$ ) to the server. This will result in a significant increase in the efficiency of the scheme, both at the server and the cloud, and both in terms of communication and computation. Communication is greatly reduced since only one customer’s information would need to be sent between the cloud and the server. Computation is also greatly reduced since the cloud does not need to prepare one randomized record and proof per customer, but only prepare them for a single customer instead.

Moreover, if **non-replication** is not an important goal for the considered application, then there is no need to compute  $A[H(G'_i, R_d) || 0^l]$  for each record during the authentication stage, which leads to another significant reduction in the computational overhead of the cloud. This comes from the fact that the cloud does not need to randomize the information that is sent to the server to prevent the server from obtaining a copy of the stored records.

Hence, if privacy and non-replication are not important goals for the use case, then our protocol becomes an  $O(1)$  solution instead of  $O(k)$ , where  $k$  denotes the number of registered customers in the database.

## VI. SECURITY

We modify existing game-based security definitions (e.g., [5]) for the four-party setting and prove security of our lattice-based construction via a formal reduction.

In developing a four-party outsourced authentication, we considered the goals in Section I-A. Out of those, we refrain from delving deep into completeness and non-replication, as those are immediately obvious. Completeness is achieved via the completeness of the Lyubashevsky’s protocol [33]. Non-replication is easily achieved by the randomization of the values, as  $R_d$  will be different at each authentication attempt, and the  $A[H(G'_j, R_d) || 0^l]$  value cannot be re-used.

For the other properties, namely soundness, confidentiality, hack-resistance, non-impersonation, and privacy, we provide two games that cover these goals.

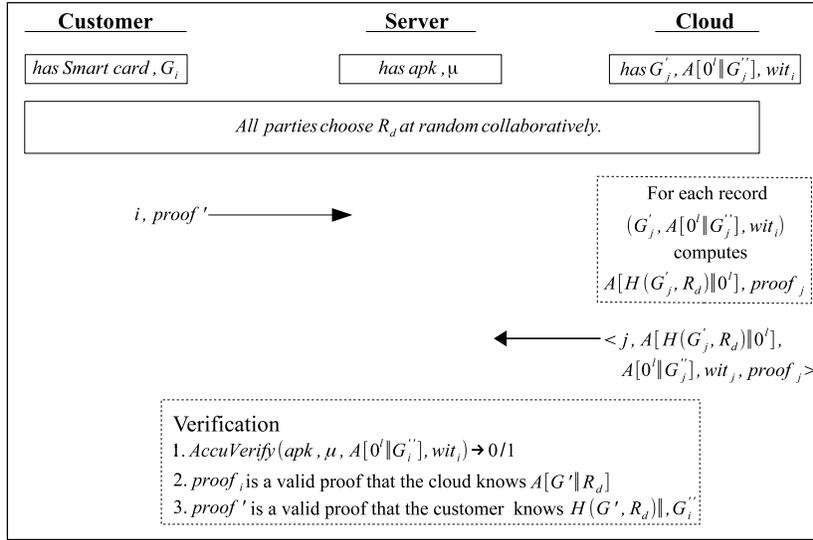


Fig. 5: Authentication stage of our Lattice-based Construction. Note that the proof can be sent by the customer immediately once  $R_d$  is agreed upon, but its verification will be done the last by the server.

The first security game is  $Out\_Aut_{S,C,HR,NI}^A(1^\lambda)$  dealing with *soundness*, *confidentiality*, *hack-resistance*, and *non-impersonation* features of a scheme. In this game, the challenger plays the roles of the honest customers, an honest server, and the trusted manager, whereas the adversary plays the roles of the cloud, dishonest servers, and an unauthorized customer.

$Out\_Aut_{S,C,HR,NI}^A(1^\lambda)$ : nolistsep

- 1) The challenger runs  $Setup(1^\lambda)$  as the manager and shares the public output with the adversary.
- 2) The challenger registers  $k$  honest customers to the database  $DB$  by running  $Register(1^\lambda, G_1, \dots, G_k)$  and sends the database  $DB$  to the adversary.
- 3) Acting as a dishonest server, the adversary may ask the challenger to authenticate with him as any honest user, polynomially-many times.
- 4) The adversary tries to authenticate with the honest server, controlled by the challenger.
- 5) The output of the experiment is “1” if the adversary succeeds, and “0” otherwise.

The advantage of the adversary  $\mathcal{A}$  in winning the game is,

$$Adv^{Out\_Aut_{S,C,HR,NI}^A}(1^\lambda) = Pr[Out\_Aut_{S,C,HR,NI}^A(1^\lambda) = 1].$$

**Definition 3.** A four-party authentication protocol provides *soundness*, *confidentiality*, *hack-resistance* and *non-impersonation* if for every probabilistic polynomial time (PPT) adversary  $\mathcal{A}$ , there is a negligible function  $negl(\cdot)$  such that,

$$Adv^{Out\_Aut_{S,C,HR,NI}^A}(1^\lambda) \leq negl(\lambda)$$

Observe that this single game covers *soundness*, *confidentiality*, *hack-resistance*, and *non-impersonation* aspects of the protocol, since it ensures that an unauthorized customer cannot authenticate with an honest server, even when the cloud is

also dishonest and helping the customer. Moreover, it also ensures that no one can impersonate an honest customer even with access to the cloud database or access to dishonest servers. Consider, for example, if the confidentiality was not provided. Then, with access to the underlying authentication data of a customer, the adversary could have authenticated with the server, thereby winning the game with non-negligible advantage. Similarly, if hack-resistance was not provided by the protocol, the adversary could have authenticated with the server, thereby winning the game with non-negligible advantage. With access to the authentication attempts of honest users against dishonest servers, if non-impersonation was not provided by the protocol, the adversary could have authenticated with the server, thereby winning the game with non-negligible advantage. Lastly, soundness is also covered since the adversary cannot authenticate with an honest server as an unauthorized customer. Therefore, this single definition is enough to cover all these security goals.

The second game  $Out\_Aut_P^A(1^\lambda)$  is related to *privacy* (including *unlinkability*), which determines whether the cloud is able to understand (or link) which user is trying to authenticate. In this game, the challenger plays the roles of the trusted manager, two honest customers, and an honest server, whereas the adversary plays the role of the cloud. Observe that since the server needs to know which customer is accessing its services, once the server and the cloud are both under the control of the adversary, no privacy can be provided. Therefore, in this game, the adversary only controls the cloud but not the server.

$Out\_Aut_P^A(1^\lambda)$ : nolistsep

- 1) The challenger runs  $Setup(1^\lambda)$  as the trusted manager and shares the public output with the adversary.
- 2) The challenger registers two customers at the database

$DB$  by running  $Register(1^\lambda, G_0, G_1)$  and gives the  $DB$  to the adversary.

- 3) The adversary may specify any customers and ask the challenger to authenticate as that customers, polynomially-many times. While the challenger internally simulates both the customer and the server, whenever the server needs data from the cloud, the challenger interacts with the adversary.
- 4) The challenger randomly chooses a bit  $b \leftarrow \{0, 1\}$  and simulates the authentication of customer  $b$  with the honest server. Again, during this authentication, whenever the server needs data from the cloud, the challenger communicates with the adversary.
- 5) Finally, the adversary outputs its guess  $b'$ .
- 6) The output of the experiment is “1” if  $b = b'$ , and “0” otherwise.

The advantage of the adversary in winning the game is,

$$Adv^{Out\_Aut_P^A}(1^\lambda) = Pr[Out\_Aut_P^A(1^\lambda) = 1] - \frac{1}{2}.$$

**Definition 4.** A four-party authentication protocol provides privacy if for every probabilistic polynomial time (PPT) adversary  $\mathcal{A}$ , there is a negligible function  $negl(\cdot)$  such that,

$$Adv^{Out\_Aut_P^A}(1^\lambda) \leq negl(\lambda)$$

We provide formal proofs that our lattice-based scheme satisfies both definitions (and hence all the desired security properties) in Appendix B.

## VII. IMPLEMENTATION

We have implemented our lattice-based scheme via the *Python 3.7* programming language and measured its efficiency. *Multiprocess (0.70.12.2)* and *Numpy (1.19.0)* are the main libraries which were employed in the implementation. The test machine had 48GB RAM and 2.1 GHz CPU with 30 cores.

In the implementation, security parameter is assumed to be  $\lambda = 192$  hence the parameters are  $n = \lambda = 192$  and  $m = \lceil 4n \log(n) \rceil = 1753$ . We employed the Fiat–Shamir heuristic and used non-interactive proofs. All records are synthesized with binary vectors of 192 bits ( $n = 192$ ). The implementation is done with different database sizes and varying number of available CPUs for all engaged entities. Running time is evaluated based on 10 executions to get average running time and avoid possible noises.

According to the obtained results, the server and the customer do not play a major role in overall running time and their running time is constant regardless of the number of records. It is observed that the customer and the server sides require only 0.3 and 0.5 seconds at most, respectively.

Database size, completeness error, and computational capacity are three factors that affect the running time of the cloud. Figure 6 depicts the total running time for the cloud during the authentication phase for different completeness error parameters and varying database sizes (between 2000 and 8000 customers).

As expected, there is a linear association between the required time for processing an authentication and the number of records in the cloud. However, it can be seen that as the number of CPU cores increases, the running time decreases. This is because the actions (randomization and preparing the proof) can be simply parallelized. With 30 CPU cores, the cloud requires one minute for 8K records. Although this running time might be high at first, it is satisfactory because of three main reasons. First, clouds normally enjoy high performance computers with more and faster CPUs compared to our system, and hence running time can be further dropped. Second, post-quantum secure cryptosystems incur higher running time in general. Therefore, this increase in running time, compared to typical classically-secure schemes, was expected. Finally, if one can accept privacy risks, (s)he can reduce the running time significantly because privacy incurs high computation and communication. To see the relation between running time and privacy, we have implemented the scheme in two cases (full privacy and no privacy) with different database sizes on a system enjoying only one CPU core. The results in Figure 7 prove that achieving privacy has great impact on running time.

This observation led us to define a tuning parameter  $\beta$  to make a balance between security and running time. This parameter is chosen from the following interval,

$$\frac{1}{|DB|} \leq \beta \leq 1.$$

Where  $|DB|$  represents the database size (the number of stored records).  $\beta$  determines the portion of records that the server should request during the authentication from the cloud.  $\beta = 1$  means all modified records are passed to the server and we have complete privacy because the cloud will not find out which user is trying to authenticate. As  $\beta$  approaches  $1/|DB|$ , we have less privacy because the cloud can observe a smaller subset of registered customers whose registration information is requested by the server, and can conclude that one of those customers is authenticating with the server. Our algorithm is implemented for different values of  $\beta$  to investigate its impact on running time and the result is exhibited in the Figure 8. Notice that  $\beta$  makes a trade-off between privacy and efficiency. As we mention in our modular architecture, without the privacy requirement (hiding which customer is authenticating from the cloud), the running time would be independent of the number of registered users, and hence would be constant.

We also investigate the communication cost in Table I. This table represents the number of elements that are transmitted. This elements belong to the chosen field over which computations are done. As it can be seen in the table, the volume of transmitted data is less than number of records multiplied by a polynomial of security parameter.

In the calculation of communication costs (Table I) constants are omitted. Note that in this table *wit*, *R* and *c.err.* represent the witness produced by accumulator algorithm, the number of records in the database and completeness

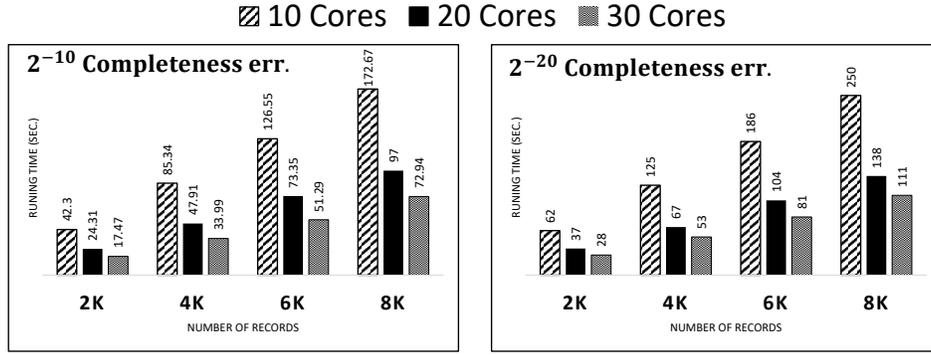


Fig. 6: The results of the implementation of lattice based scheme over four synthetic databases with 2k, 4k, 6k and 8k records with  $2^{-10}$  and  $2^{-20}$  completeness error. X-Axis represents the database’s size stored in the cloud and the Y-Axis shows running in seconds. Furthermore, the implementation is done with a different number of assigned CPUs to detect its influence on running time.

TABLE I: Communication overhead of the proposed scheme.

	$2^{-10}$ c. err.	$2^{-20}$ c. err.
<b>Reg.</b>	$R(o(n^2) + wit)$	$R(o(n^2) + wit)$
<b>Auth.</b>	$1200n \log n + \beta R(1208n \log n + wit)$	$2400n \log n + \beta R(2408n \log n + wit)$

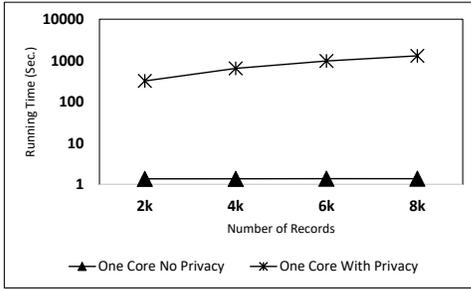


Fig. 7: Measuring the impact of privacy on running time. Only one CPU is leveraged and implementation is done over databases with different number of records. X-Axis represents the number of registered persons and Y-Axis shows the running time of authentication process.

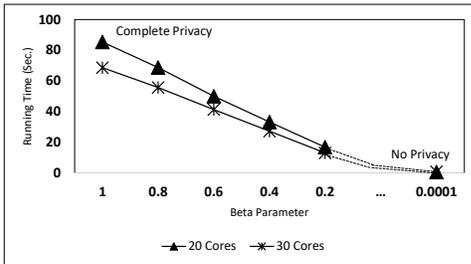


Fig. 8: Cloud running time with a database of size 10k for different values of  $\beta$  and Completeness error equals to  $2^{-10}$ . X-axis and Y-axis indicate  $\beta$  and running time, respectively.

error, respectively. Cost  $wit$  varies according to the adopted accumulator algorithm. It, however, is a polynomial factor. As it can be seen in the table during registration stage what is transmitted depends on  $n$  and the database size has no

impact. In the authentication stage, the transmitted volume is determined by completeness error,  $n$ ,  $\beta$  and the number of the records. In the proposed scheme,  $\beta$  can be chosen according to the user’s policies to reduce computation overhead at the cost of increasing privacy risk.

Table ?? provides an example of the volume of transmitted messages required for the authentication process in three databases with different sizes. The registration is excluded because it only depends on the security parameter but not the number of records. The security parameter is equal to 128. Furthermore, a lattice-based accumulator in [22] is used.

To the best of our knowledge, there is no paper that has a similar four-party authentication model. Therefore, we decided to include [39] to benchmark the proposed scheme with it in view of communication costs. Note that, their scheme does not support privacy. Hence, we added the communication costs in our scheme when privacy is not preserved ( $\beta = 1/|DB|$ ). However, it can be seen that there is a slight difference, which comes from the fact that in our scheme authentication data is outsourced and servers get randomized authentication data which means “one-time registration but multiple servers authentication” in contrast to Li et al. [39].

## VIII. CONCLUSION AND FUTURE WORK

Authentication models are changing with the advent of cloud. In this paper, we define three- and four-party authentication scenarios, where the customer database is not at the login server but at the cloud, and hence novel approaches are required.

We provided generic methods to convert existing secure two-party authentication protocols to the three- and four-party model. We defined several threat models and security objectives, including *completeness*, *soundness*, *confidentiality*,

TABLE II: Communication overhead in a real examples. In this table security parameter is 128 except for *no privacy* versions (gray rows) where the security parameter is 752. The derived values are in megabytes (MB). The errors  $e_1$  and  $e_2$  denote  $2^{-10}$  and  $2^{-20}$  completeness errors, respectively.

	2k		4k		6k	
	$e_1$	$e_2$	$e_1$	$e_2$	$e_1$	$e_2$
$\beta = 0.1$	146.845	292.724	292.966	583.998	585.206	1166.545
$\beta = 0.05$	73.785	147.088	146.845	292.724	292.966	583.998
$\beta = 0.01$	15.337	30.578	29.949	59.706	59.173	117.960
No Privacy	10.122	20.206	10.122	20.206	10.122	20.206
Li <i>etal.</i> [39]	8.535	8.535	8.535	8.535	8.535	8.535

*hack-resistance, non-impersonation, privacy* (including *unlinkability*), and *non-replication*, and discussed how to achieve them under different settings. Moreover, we provided a special lattice-based post-quantum secure construction with full details and security proof in the four-party model. We leave achieving security against a malicious cloud provider as future work.

The data that support the findings of this study are not openly available because the database is generated randomly and might not be useful for the readers. However, it is available from the corresponding author upon reasonable request.

#### REFERENCES

- [1] P. Peer, J. Bule, J. Ž. Gros, and V. Štruc, “Building cloud-based biometric services,” *Informatica*, vol. 37, no. 2, 2013.
- [2] H. Chun, Y. Elmehdwi, F. Li, P. Bhattacharya, and W. Jiang, “Outsourceable two-party privacy-preserving biometric authentication,” in *ACM CCS*. ACM, 2014, pp. 401–412.
- [3] M. Haghghat, S. Zonouz, and M. Abdel-Mottaleb, “Cloudid: Trustworthy cloud-based and cross-enterprise biometric identification,” *Expert Systems with Applications*, vol. 42, no. 21, pp. 7905–7916, 2015.
- [4] C. Hahn and J. Hur, “Efficient and privacy-preserving biometric identification in cloud,” *ICT Express*, vol. 2, no. 3, pp. 135–139, 2016.
- [5] T. Acar, M. Belenkiy, and A. K p c , “Single password authentication,” *Computer Networks*, vol. 57, no. 13, pp. 2597–2614, 2013.
- [6] D.  şler and A. K p c , “Threshold single password authentication,” in *ESORICS DPM*. Springer, 2017, pp. 143–162.
- [7] W. Ford and J. Burton S. Kaliski, “Server-assisted generation of a strong secret from a password,” in *WETICE*, 2000.
- [8] P. D. MacKenzie, T. Shrimpton, and M. Jakobsson, “Threshold password-authenticated key exchange,” in *CRYPTO*, 2002.
- [9] D. Florencio and C. Herley, “A large-scale study of web password habits,” in *WWW*. ACM, 2007, pp. 657–666.
- [10] M. Blanton and P. Gasti, “Secure and efficient protocols for iris and fingerprint identification,” in *ESORICS*. Springer, 2011, pp. 190–209.
- [11] J. Bringer, H. Chabanne, and B. Kindarji, “Identification with encrypted biometric data,” *Security and Communication Networks*, vol. 4, no. 5, pp. 548–562, 2011.
- [12] Z. Erkin, M. Franz, J. Guajardo, S. Katzenbeisser, I. Lagendijk, and T. Toft, “Privacy-preserving face recognition,” in *PETS*. Springer, 2009, pp. 235–253.
- [13] H. Gao, Y. Zhang, S. Liang, and D. Li, “A new chaotic algorithm for image encryption,” *Chaos, Solitons & Fractals*, vol. 29, no. 2, pp. 393–399, 2006.
- [14] E.-J. Goh, “Secure indexes,” *IACR Cryptology ePrint Archive*, vol. 2003, p. 216, 2003.
- [15] A. K. Jain, A. Ross, and U. Uludag, “Biometric template security: Challenges and solutions,” in *EUSIPCO*. IEEE, 2005, pp. 1–4.
- [16] M. Osadchy, B. Pinkas, A. Jarrous, and B. Moskovich, “Scifi-a system for secure face identification,” in *IEEE Symposium on Security and Privacy*. IEEE, 2010, pp. 239–254.
- [17] M. Upmanyu, A. M. Namboodiri, K. Srinathan, and C. Jawahar, “Blind authentication: a secure cryptobiometric verification protocol,” *IEEE Transactions on Information Forensics and Security*, vol. 5, no. 2, pp. 255–268, 2010.
- [18] S. M. Bellovin and M. Merritt, “Encrypted key exchange: Password-based protocols secure against dictionary attacks,” in *IEEE Symposium on Security and Privacy*, 1992.
- [19] —, “Augmented encrypted key exchange: A password-based protocol secure against dictionary attacks and password file compromise,” in *ACM CCS*, 1993.
- [20] X. Boyen, “Hpake: Password authentication secure against cross-site user impersonation,” in *CANS*, 2009.
- [21] J. Benaloh and M. De Mare, “One-way accumulators: A decentralized alternative to digital signatures,” in *EUROCRYPT*. Springer, 1993, pp. 274–285.
- [22] J. Li, N. Li, and R. Xue, “Universal accumulators with efficient nonmembership proofs,” in *ACNS*. Springer, 2007, pp. 253–269.

- [23] J. Camenisch and A. Lysyanskaya, “Dynamic accumulators and application to efficient revocation of anonymous credentials,” in *CRYPTO*. Springer, 2002, pp. 61–76.
- [24] L. Nguyen, “Accumulators from bilinear pairings and applications,” in *CT-RSA*. Springer, 2005, pp. 275–292.
- [25] M. O. Rabin, “How to exchange secrets with oblivious transfer,” *IACR Cryptology ePrint Archive*, vol. 2005, p. 187, 2005.
- [26] C. Crépeau, “Equivalence between two flavours of oblivious transfers,” in *CRYPTO*. Springer, 1987, pp. 350–354.
- [27] W. Ogata and K. Kurosawa, “Oblivious keyword search,” *Journal of Complexity*, vol. 20, no. 2-3, pp. 356–371, 2004.
- [28] A. Shamir, “How to share a secret,” *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.
- [29] G. R. Blakley, “Safeguarding cryptographic keys,” in *International Workshop on Managing Requirements Knowledge*. IEEE Computer Society, 1979, pp. 313–313.
- [30] M. Ajtai, “Generating hard instances of lattice problems,” in *ACM STOC*. ACM, 1996, pp. 99–108.
- [31] P. W. Shor, “Algorithms for quantum computation: Discrete logarithms and factoring,” in *IEEE FOCS*. IEEE, 1994, pp. 124–134.
- [32] D. Micciancio and S. Goldwasser, *Complexity of lattice problems: a cryptographic perspective*. Springer Science & Business Media, 2012, vol. 671.
- [33] V. Lyubashevsky, “Lattice-based identification schemes secure under active attacks,” in *PKC*. Springer, 2008, pp. 162–179.
- [34] M. Blum, “Coin flipping by telephone a protocol for solving impossible problems,” *ACM SIGACT News*, vol. 15, no. 1, pp. 23–27, 1983.
- [35] B. Alon and E. Omri, “Almost-optimally fair multiparty coin-tossing with nearly three-quarters malicious,” in *TCC*. Springer, 2016, pp. 307–335.
- [36] I. Haitner, N. Makriyannis, and E. Omri, “On the complexity of fair coin flipping,” in *TCC*. Springer, 2018, pp. 539–562.
- [37] I. Damgård, “On sigma protocols,” <http://www.daimi.au.dk/~ivan/Sigma.pdf>.
- [38] A. Fiat and A. Shamir, “How to prove yourself: Practical solutions to identification and signature problems,” in *CRYPTO*. Springer, 1986, pp. 186–194.
- [39] Z. Li, D. Wang, and E. Morais, “Quantum-safe round-optimal password authentication for mobile devices,” *IEEE Transactions on Dependable and Secure Computing*, 2020.

### A. Lyubashevsky’s lattice based authentication scheme

Lyubashevsky proved that if we repeat the protocol in the table  $t = \omega(\log(n))$  times in parallel, then the completeness error is less than  $2^{-\frac{t}{14}}$ . The following theorem indicates the connection between the identification protocol’s robustness and solving the SIS problem.

**Theorem 2.** [33] *If there is a probabilistic polynomial time (PPT) adversary which can break the above identification protocol with probability  $adv$  in the active attack model, then there exists a polynomial algorithm that can solve the SIS(A) problem with success probability  $\omega((adv)^2 - 2 \cdot 2^{-\frac{t}{15}})$  when  $A$  is chosen uniformly at random.*

### B. Security Proofs

In this section we prove that our lattice-based construction meets the considered aims.

**Theorem 3.** *If the SIVP problem is hard, then no PPT adversary can win the  $Out\_Aut_{S,C,HR,NI}$  game with non-negligible advantage.*

*Proof.* We prove this theorem by showing that the SIVP can be solved with a polynomial factor given a PPT adversary who wins the  $Out\_Aut_{S,C,HR,NI}^A(1^\lambda)$  game with non-negligible advantage  $\epsilon(\lambda)$ .

Assume that there exists a PPT adversary  $\mathcal{A}$  who wins the game with non-negligible advantage  $\epsilon(\lambda)$ . Then, we show a PPT algorithm  $\mathcal{B}$  that can be built based on  $\mathcal{A}$  such that  $\mathcal{B}$  solves the SIS problem for randomly chosen matrix  $A \in \mathbb{Z}_p^{n \times m}$  where  $m = \lceil 4n \log n \rceil$  and  $n = \lambda$ .

For the security reduction, the algorithm  $\mathcal{B}$  gets a random matrix  $A \in GL(n, m)$  and must produce two vectors  $z$  and  $z'$  with norms less than  $10m^{1.5}$  such that  $Az = Az'$ .

Given the random matrix  $A$ , the algorithm  $\mathcal{B}$  works as follows,

- 1)  $\mathcal{B}$  needs to simulate  $Setup(1^\lambda)$  as the manager.  $\mathcal{B}$  uses the given matrix  $A$ , and picks a collision-resistant hash function  $H$ .
- 2)  $\mathcal{B}$  chooses  $k \in poly(\lambda)$  binary vectors  $G_1, \dots, G_k \in \{0, 1\}^m$  randomly and, with the *Register* algorithm, acting as the challenger in  $Out\_Aut_{S,C,HR,NI}^A(1^\lambda)$ , builds a database  $DB$  in which its rows are  $(G'_i, A[0^i || G'_i], wit_i)$ . In addition to the  $DB$ , the accumulator keys  $(apk, ask)$  and the digest  $\mu$  of these records are generated just as the honest challenger would have done.
- 3)  $\mathcal{B}$  shares the public values  $H, DB, A, apk$ , and  $\mu$  with the adversary  $\mathcal{A}$ . Note that the adversary, as the cloud, needs to know  $H, DB, A$ , and the adversary, as the dishonest servers, needs to know  $apk$  and  $\mu$ .
- 4) Upon request of the adversary  $\mathcal{A}$ ,  $\mathcal{B}$  runs the authentication protocol with dishonest servers just as an honest challenger would have.

Prover	Verifier
Private key: $\tilde{\omega} \in \{0, 1\}^m$ Public key: $A, \omega = A\tilde{\omega} \pmod p$	
for $i = 1$ to $t$ $\tilde{y}_i \xleftarrow{\$} \{0, \dots, 5m - 1\}^m$ $y_i \leftarrow A\tilde{y}_i \pmod p$	
	$\xrightarrow{y_1, y_2, \dots, y_t}$
	for $i = 1$ to $t$ $c_i \xleftarrow{\$} \{0, 1\}$
	$\xleftarrow{c_1, \dots, c_t}$
for $i = 1$ to $t$ if $c_i = 1$ and $\tilde{y}_i + \tilde{\omega} \notin SAFE$ $z_i \leftarrow \perp$ else $z \leftarrow \tilde{y}_i + c_i \tilde{\omega}$	
	$\xrightarrow{z_1, \dots, z_t}$
	for $i = 1$ to $t$ if $\ z_i\  \leq 5m^{1.5}$ and $Az_i = c_i \omega + y_i$ $d_i \leftarrow 1$ else $d_i \leftarrow 0$ $sum = d_1 + \dots + d_t$
	if $sum \geq 0.65t$ then ACCEPT else REJECT

TABLE III: One round of the identification protocol of [33]. In this scheme  $p$  is an integer of order  $\tilde{O}(n^3)$ ,  $m = \lceil 4n \log n \rceil$ , and  $SAFE = \{1, \dots, 5m - 1\}^m$ .

- 5)  $\mathcal{B}$  plays the role of server and interacts with the adversary. In this stage, the adversary acting as the customer specifies  $i_{adv}$  and sends it to  $\mathcal{B}$ .
- 6)  $\mathcal{B}$  interacts with the cloud, which is controlled by the adversary and receives  $A\omega, A[0^l || G''_{i_{adv}}], wit_{i_{adv}}, proof_{i_{adv}}$ . For the adversary to be successful, the witness should be valid according to  $AccuVerify(apk, \mu, A[0^l || G''_{i_{adv}}], wit_{i_{adv}})$  and the proof  $proof_{i_{adv}}$  must verify.
- 7) Then, the adversary, acting as the customer, should prove that it knows a binary vector  $\omega'$  such that,

$$A\omega' = A[\omega + 0^l || G''_{i_{adv}}] \quad (1)$$

Thus, there are two proofs that the adversary must perform successfully: one as the cloud, and one as the customer.

We have  $Adv^{Out\_Aut_{S,C,HR,NI}^A}(1^\lambda) = \epsilon(\lambda)$ . In addition, we have:

$$\begin{aligned}
 Adv^{Out\_Aut_{S,C,HR,NI}^A}(1^\lambda) &= \\
 &Pr[Suc|\omega, \neg\omega']Pr[\omega, \neg\omega'] + Pr[Suc|\neg\omega, \neg\omega']Pr[\neg\omega, \neg\omega'] \\
 &+ Pr[Suc|\neg\omega, \omega']Pr[\neg\omega, \omega'] + Pr[Suc|\omega, \omega']Pr[\omega, \omega'] \\
 &\leq Pr[Suc|\omega, \neg\omega'] + Pr[Suc|\neg\omega, \neg\omega'] + Pr[Suc|\neg\omega, \omega'] \\
 &+ Pr[\omega, \omega']
 \end{aligned}$$

where, for instance,  $Pr[Suc|\neg\omega, \omega']$  means the success probability of the adversary in proving the two statements while it knows  $\omega'$  but does not know  $\omega$ , and  $Pr[\neg\omega, \omega']$  is the

probability that the adversary knows  $\omega'$  but does not know  $\omega$ . The other terms are defined similarly. To bound this probability, simply realize that each probability is between zero and one.

It can be demonstrated that  $Pr[Suc|\omega, \neg\omega']$ ,  $Pr[Suc|\neg\omega, \omega']$  and  $Pr[Suc|\neg\omega, \neg\omega']$  are negligible. For example,  $Pr[Suc|\omega, \neg\omega']$  is the success probability of the adversary passing both proof verifications having access to only  $\omega$  but not  $\omega'$ . As mentioned earlier, the adversary should prove the two statements. In the second one, it needs to prove that it has access to a binary sequence  $\omega'$  such that  $A\omega' = A[\omega + 0^l || G''_{i_{adv}}]$ . Due to the fact that the soundness probability of Lyubashevsky's scheme is negligible, the probability that the adversary can convince  $\mathcal{B}$  while it does not have  $\omega'$  is negligible. Accordingly, it can be deduced that  $Pr[\omega, \omega']$  is non-negligible because,

$$\begin{aligned}
 Pr[\omega, \omega'] &\geq Adv^{Out\_Aut_{S,C,HR,NI}^A}(1^\lambda) \\
 &- (Pr[Suc|\omega, \neg\omega'] + Pr[Suc|\neg\omega, \neg\omega'] + Pr[Suc|\neg\omega, \omega']) \\
 &\geq \epsilon(\lambda) - negligible(\lambda)
 \end{aligned}$$

This implies that with a non-negligible probability, whenever the adversary wins the game, it has access to the binary values  $\omega'$  and  $\omega$ . Hence,  $\mathcal{B}$  is able to extract these values from the proofs using the knowledge extractor, in probabilistic polynomial time. In addition,  $\omega' - \omega \in \{-1, 0, 1\}^m$  and  $A(\omega' - \omega) = A[0^l || G''_{i_{adv}}]$ . Furthermore,  $\|\omega' - \omega\|, \|(0^l || G''_{i_{adv}})\| \leq$

$m \leq 10m^{1.5}$  because their components belong to  $\{-1, 0, 1\}$ . Therefore,  $z = \omega' - \omega$  and  $z' = 0^l || G''_{i_{adv}}$  are a solution for  $SIS(A)$ . Finally, according to Theorem 1, the ability to solve the  $SIS$  problem leads to solving  $SIVP$ . Consequently, breaking the proposed scheme with non-negligible advantage leads to a PPT algorithm which can solve  $SIVP$  with a polynomial approximation factor.  $\square$

**Theorem 4.** *No PPT adversary can win the  $Out\_Aut_P$  game with non-negligible advantage.*

*Proof.* In order to prove that any PPT adversary cannot guess better than a random guess, we will show that the information the cloud gets regarding both customers are computationally indistinguishable. The adversarial cloud is only involved in two steps. First, in choosing the random value  $R_d$ , which is completely independent of stored records in the database. Secondly, when the server tries to retrieve required information used in the authentication process, it requests all customers' records, and hence this again does not lead to a guess noticeably better than a random guess.  $\square$

As discussed before, *completeness* is achieved via the completeness of the Lyubashevsky's protocol [33]. *Non-replication* is achieved by the randomization of the values, as  $R_d$  will be different at each authentication attempt, and the  $A[H(G'_j, R_d) || 0^l]$  value cannot be re-used. Picking a random  $R_d$  each time also helps achieving *non-impersonation*, as it was already covered in Theorem 3. Note that at each authentication attempt, at least one participant is honest and contributes to the randomness. When the adversary is a malicious customer and a semi-honest cloud, then the server is honest. When the adversary is a semi-honest server and the semi-honest cloud, then the customer is honest.