Distributed Certified Information Access for Mobile Devices*

Aniello Del Sorbo[1], Clemente Galdi[2], and Giuseppe Persiano[1]

[1] Dipartimento di Informatica ed Applicazioni "R.M. Capocelli"
Università degli Studi di Salerno
Via Ponte Don Melillo - 84084 Fisciano (SA) - Italy
{anidel,giuper}@dia.unisa.it
[2] Dipartimento di Scienze Fisiche
Università degli Studi di Napoli "Federico II"
Complesso Univ. Monte S. Angelo - Via Cinthia
80126 Napoli - Italy
galdi@na.infn.it

Abstract. In this paper we describe a primitive, which we call, Certified Information Access, in which a database answers to a query by providing the information matching the query along with a proof that such information are consistent with the actual content of the database. We show that such a primitive can be securely implemented in a distributed fashion. Furthermore, we describe the design principles for a distributed architecture that would allow the use of this primitive on mobile devices.

Key words: Secure protocols; Secure services for mobile devices.

1 Introduction

The growing need of mobility in the current society and the increasing availability of low-cost wireless devices have fostered an impressive growth in the number of services available for such devices. Currently available technologies allow the possibility of performing tasks on wireless devices that were impossible only few years ago. Theoretically, it is now possible to run any Java program on last generation mobile devices. On one hand, this allows the possibility of interaction between any Java-enabled mobile device with any possible application that supports web-based access. It is possible to download programs from the Internet and execute them in order to interact with a specific service.

This flexibility poses a number of security issues that need to be addressed. Just to mention a few, authentication, anonymity, accountability of users/services need be to thought for a environment in which the device has very low computational abilities, the communication medium can be easily eavesdropped or subject to malicious attacks of various kind, etc. In particular, because of the small

* This work was partially supported by the European Union under IST FET Integrated Project AEOLUS (IST-015964).

computational power of such devices, one issue to address is the performance of the applications.

In a such global scenario it is often the case that one entity has to access information owned by a different entity. This poses a number of security issues both from the database side and the user side. For example, the owner of the database may require that only authorized users have access to the information or part of it. This is an instance of the well-known access control problem. Another example could be that each user cannot infer additional knowledge by analyzing the answers to the issued queries. In other words, authorized users can obtain the information they required, but they cannot infer any other information from the received answers [5, 9]. On the other hand, a user may require that the database does not gain any information about specific content she requested [6, 4, 10, 11, 1]. This means that the database should "blindly" answer all the queries from authorized users in a way that all the requested information can be correctly reconstructed from the answers.

The above problems mainly address the confidentiality of the information. The primitive we consider in this prototype, introduced in [8], deals with the problem of guaranteeing the consistency of the answer, sent by the database to the user, with the information actually contained in the database. We assume the possibility that a database would be willing to give wrong answer to queries issued by a user. In this setting, since the user does not know in advance which is the actual information he will receive from the database, there would be no way to distinguish between a correct answer from a maliciously modified one.

Certified Information Access (CIA for short) primitives force the database to publish a snapshot of its current contents, which we refer to as the *Public Information*, on a trusted entity. After such information is available, the user may issue queries to the database. The answers to each query have to be *consistent* with the public information.

One trivial way of implementing such primitives is to publish the whole content of the database on a trusted server. In this case, a user may compare the received query with the one contained in the public copy of the database. This solution is, of course, neither secure nor efficient. Since the database has to publish its whole contents, the confidentiality of the information therein contained is compromised. Furthermore, since all the elements in the database need to be transmitted and stored, the communication and space complexities of such solution is linear in the size of the database

For this reasons the public information should satisfy the following properties:

- *Compactness:* The size of the public information should be smaller than the size of the database.
- *Confidentiality:* The public information should not reveal anything about the actual content of the database.
- *Correctness:* A correct answer to a query should be consistent with the public information with probability one.
- *Soundness:* Any wrong answer to a query will be detected with high probability.

Currently, a way of implementing CIA primitives is by means of a new cryptographic primitive, namely *mercurial commitments* introduced and studied in [8, 3, 2, 7]. Unfortunately, the implementation of such primitive are computationally intensive. On one hand, the generation of the public information is time consuming also on current servers. On the other hand, although the verification procedure can be easily executed on a PC in few seconds, it still requires much more time on mobile devices.

Our Contribution. In this paper we present a distributed architecture for a CIA service. We first describe in details the Certified Information Access service. We briefly review the basic primitives that can be used for implementing such a service. However, such primitives require a certain amount of computation that would make any solution for CIA unfeasible on mobile devices or, more generally, on devices with low computational power. We show that such primitives can be computed securely in a distributed fashion. In other words, any device can distribute its load securely among untrusted *peers* and locally combine the results of such computations. We finally show an architectural design for the distributed implementation of a secure CIA service.

We describe a solution for *static* databases, i.e., databases in which the content does not change. To the best of our knowledge, the *only* secure solution to the problem of dynamic databases is the one described in [7]. Unfortunately this solution is not efficient. Indeed, updating a single element in the database results in the need of storing some information whose size is linear in the size of the key.

We assume that there exists a trusted entity that does not collude with the entity holding the database. Furthermore, the system comprises a sufficient number of peers whose only role is to compute modular exponentiations. We assume that such peers are honest, that is, they properly execute the protocols, but a small fraction of them may be curious, in the sense that they may collude in order to infer additional information from the messages they have exchanged.

This document is organized as follows: In Section 2 we describe the CIA primitive. In Section 3 we describe a basic tool that is used in the prototype, namely Mercurial Commitment schemes, and techniques for distributing the computation of such primitive among the peers. In Section 4 we describe how to construct a certified information access primitives using mercurial commitments. In Section 5 we report the design principles for a distributed architecture implementing a CIA service.

2 Certified information access

In this section we describe the CIA functionality that we will be at the base of our prototype.

In the context of secure databases, an implementation of a certified information access has to provide the users with a database service in which each answer to a query consists of the actual query results and a proof that such information

is indeed the actual content of the database. The verification of the proof can be accomplished by using some public information that the database provided before the query was issued. Such public information should not reveal anything about the actual content of the database. In a CIA system we identify three parties, the CertifiedDBOwner the User and the PubInfoStorage.

In a setup phase, the PubInfoStorage generates the public parameters that will be used for the CIA service. The PubInfoStorage is assumed not to collude with the CertifiedDBOwner.

The CertifiedDBOwner, based on public parameters and the content of the database, produces the public information that is then sent to the PubInfoStorage. Whenever a User makes a query to the CertifiedDBOwner, he obtains an object that contains the answer to the query and some information that can be used, along with the information held by the PubInfoStorage, to prove that the answer is indeed correct and that the CertifiedDBOwner has not cheated.

Cryptographic background. A very simple type of Certified Information Access, is the one in which the database consists of only one string, one-string CIA (or 1-CIA). The one-string CIA functionality has been studied in Cryptography under the name of *commitment* and several implementations of this primitive have been presented. Instead, the concept of a M1-CIA corresponds to a special type of commitments called *mercurial* commitments introduced by [3] and later studied by [2, 7].

1-CIA can be seen as a safe. The CertifiedDBOwner writes the string m on a piece of paper, puts it in a safe and locks the safe. The safe can be sent to a User that cannot open it (thus guaranteeing the *hiding* property). On the other hand the User is guaranteed that the message in the safe cannot change while it is in the safe (thus guaranteeing the *binding* property). Whenever the CertifiedDBOwner chooses to, he can *open* the safe by sending the string m and the key to open the safe. The User can then *verify* that the value he sees is the same as the original message stored in the safe.

Using a commitment scheme, we can implement a 1-CIA as follows: the CertifiedDBOwner, based on the actual value m of the string, produces a *commitment* and a *decommitment key*. The commitment is sent to the PubInfoStorage. Whenever the CertifiedDBOwner chooses to, he *opens* the commitment by releasing the the value of m and the decommitment key. The User can verify that the opening has been correctly performed by checking the open against the information held by the PubInfoStorage.

The M1-CIA functionality is an extension of the 1-CIA functionality with an important extra property. In addition to the usual operation of opening a commitment, M1-CIA also supports a *partial open* operation called *tease*. A commitment `com` can be computed in two ways: it can be a *hard* commitment, that is a commitment that can be opened and teased in only one way; or a *soft* commitment that cannot be opened at all, but can be teased to any value.

The *binding* and *hiding* properties also hold for a M1-CIA. In addition, hard commitments are indistinguishable from soft ones. In particular, this means that

it is computationally infeasible to distinguish whether a commitment com is a soft or a hard one.

The mechanism is the same as the ones of the 1-CIA. The only difference is that the CertifiedDBOwner can also produce soft commitment that can be teased to any string m.

A M1-CIA scheme provides the following functions.

CertifiedDBOwner.

- Commit: computes, on input the string m and the public parameters, the *commitment* com to be sent to PubInfoStorage and the *decommitment key* dec to be used in the opening.
- SoftCommit: computes, on input the public parameters, a *soft commitment* Scom along with a *teasing key* Sdec to be used for teasing Scom.
  Notice that SoftCommit does not need a string m as Scom can be teased to any value m.
- Tease: computes, on input the public parameters, a string m, a commitment com (com could be a hard or a soft commitment) and a teasing key Sdec, the *teasing* τ of Scom to string m.

User.

- VerifyOpen: Given the public parameters, verifies that message m and a decommitment key dec, are consistent with a commitment com.
- VerifyTease: verifies, on input the public parameters, that a teasing τ of a commitment com (it could be a soft commitment or a hard one) to a string m has been correctly computed.

3 Primitives

In this section we show how we implement the M1-CIA functionality. Our implementation is based on the hardness of the discrete logarithm in cyclic groups and is based on [3].

3.1 Implementing M1-CIA via Mercurial Commitments

The Setup procedure (executed by the PubInfoStorage) consists in randomly picking a random prime p and two generators g, h of the cyclic group $Z_p^\star$. All operations are to be considered in the group $Z_p^\star$ unless otherwise specified.

The Commit procedure (executed by the CertifiedDBOwner) takes as input the string m and public parameters (p, g, h) and computes com and dec as follows: randomly pick $r_0, r_1 \in Z_{p-1}^\star$ and set $\texttt{com} = (g^m \cdot (h^{r_1})^{r_0}, h^{r_1})$ and $\texttt{dec} = (r_0, r_1)$.

The VerifyOpen procedure (executed by the User) takes as input the public parameters (p, g, h), a message m, a commitment $\texttt{com} = (C_0, C_1)$ and

decommitment key $\texttt{dec} = (r_0, r_1)$ and consists in checking that $C_0 = g^m \cdot C_1^{r_0}$ and $C_1 = h^{r_1}$.

The SoftCommit procedure (executed by the CertifiedDBOwner) takes as input the public parameters (p, g, h) and computes $\texttt{Scom}$ and $\texttt{Sdec}$ as follows: randomly pick $r_0, r_1 \in Z^{\star}_{p-1}$ and set $\texttt{Scom} = (g^{r_0}, g^{r_1})$ and $\texttt{Sdec} = (r_0, r_1)$.

The teasing τ of a hard commitment $\texttt{com} = (g^m \cdot (h^{r_1})^{r_0}, h^{r_1})$ of string m with the decommitment key $\texttt{dec} = (r_0, r_1)$ consists simply of $\tau = r_0$.

Instead the teasing τ of a soft commitment $\texttt{Scom} = (g^{r_0}, g^{r_1})$ with teasing key $\texttt{Sdec} = (r_0, r_1)$ to string m is computed by setting $\tau = (r_0 - m)/r_1 \pmod{p-1}$.

The VerifyTease procedure (executed by the User) takes as input public parameters (p, g, h) and teasing τ of commitment (C_0, C_1) to string m consists in checking that $C_0 = g^m \cdot C_1^{\tau}$.

Correctness and security of this scheme have been shown in [3].

3.2 Distributing M1-CIA Computation

This paragraph describes a way of distributing the computations needed to create and verify mercurial commitments while preserving the security of the M1-CIA scheme.

Since the most time-consuming operation is the modular exponentiation, we show a way of distributing such operation securely. The idea behind the load distribution is to use the computational power of peers to execute modular exponentiations. In this way the CertifiedDBOwner and the User are only required to execute additions and multiplications.

Secure computation of exponentiations Crucial operations to be distributed are modular exponentiations in which either the exponent or both the base and the exponent are sensitive information. We assume that a service Mod_Exp is run a set of peers that the User and the CertifiedDBOwner may use for such operations. Peers are assumed to be honest, i.e., compute correctly the modular exponentiations they are required to, but curious, in the sense that they may collect the information received in order to obtain information on the values queried by the user or on the elements of the database.

Mod_Exp*:* We assume that the peers can be identified by an ID in the set $\{1, \ldots, t\}$, for some integer t. This service is inkoved with input a base b, an exponent r, the modulus p and the ID of the peer that will execute the task. The peer with identity ID computes $b^r \bmod p$ and sends back the result to the player who required it. We assume secure point-to-point communication between peers and the player who invokes their services.

Computation with secure exponent. We first analyze the case in which the exponent is a secret information. Let k be an integer such that $k - 1 \ll t$. We assume that the maximum number of colluding peers is at most $k - 1$. In this case, given b and e, it is possible to compute $b^e \bmod p$ keeping the exponent e private as follows:

Procedure SECURE_EXP(b, e, p, k)

- randomly select k out of the t peers and let $\{ID_1, \ldots, ID_k\}$ be their identities.
- pick $e_1, \ldots, e_{k-1} \in_R Z_{p-1}$
- $e_k = e - (e_1 + \ldots + e_{k-1}) \bmod (p-1)$.
- $r_i = \text{MOD\_EXP}(b, e_i, p, ID_i)$, for $i = 1, \ldots, k$.
- $r = \prod_{i=1}^{k} r_i \bmod p$
- output r

The correctness of the above procedure follows immediately from the fact that $r_i = b^{e_i} \bmod p$ and, thus, $r = b^{e_1+\ldots+e_k} = b^e \bmod p$. Security follows from the observation that the exponent e is shared among the k peers using a (k,k)-threshold secret sharing scheme. Thus, the only way to reconstruct e is to collect all the k shares.

Computation with secure base and exponent. Using a similar idea, it is possible to compute b^e while keeping both the base b and the exponent e private. The main difference is that, in this case, we need to properly share both the base and the exponent and recombine the partial results.

Procedure SECURE_BASE_EXP(b, e, p, k^2)

- randomly select k^2 peers out of the t and let $\{ID_1, \ldots, ID_{k^2}\}$ be their identifiers
- pick $e_1, \ldots, e_{k-1} \in_R Z_{p-1}$
- pick $b_1, \ldots, b_{k-1} \in_R Z_p^\star$
- $e_k = e - (e_1 + \ldots + e_{k-1}) \bmod (p-1)$.
- $b_k = b / \prod_{i=1}^{k} b_i \bmod p$.
- $r_{i,j} = \text{MOD\_EXP}(b_i, e_j, p, ID_{i(j-1)+j-1})$, for $i = 1, \ldots, k$ and $j = 1, \ldots, k$
- $r_i = \prod_{j=1}^{k} r_{i,j} \bmod p$
- $r = \prod_{i=1}^{k} r_i \bmod p$
- output r

The correctness of the above procedure can be derived by observing that $r_i = b_i^e \bmod p$ since SECURE_BASE_EXP implicitly contains an invocation of the procedure SECURE_EXP with parameters b_i and e. Furthermore, $r = b_1^e \cdot \ldots \cdot b_k^e = (\prod_{i=1}^{k} b_i)^e = b^e \bmod p$. The security of the procedure derives from the fact that we use two independent (k,k)-threshold secret sharing schemes for sharing b and e. Since each pair (b_i, e_j), for $i, j \in \{1, \ldots, k\}$, is assigned to a different peer, the only way to reconstruct the value of b (resp., the value of e) is to collect all the values b_i (resp., e_i).

Distributing the commitment operations. Given the above procedures, we can distribute the computation of commitments as follows:
Recall that, given the public parameters (p, g, h), a soft commitment consists of a pair $\texttt{Scom} = (g^{r_0}, g^{r_1})$ and $\texttt{Sdec} = (r_0, r_1)$, where r_0 and r_1 are randomly chosen.

In this case, the values r_0 and r_1 need to be kept private since they are used for the teasing of $\texttt{Scom}$. Thus, the SoftCommit procedure can be distributed as follows:

Procedure SoftCommit(p, g, h, k)

- $r_0, r_1 \in_R Z_p$
- $y_0 = \text{Secure\_Exp}(g, r_0, p, k)$
- $y_1 = \text{Secure\_Exp}(g, r_1, p, k)$
- output $\texttt{Scom} = (y_0, y_1)$, and $\texttt{Sdec} = (r_0, r_1)$.

The correctness of the above procedure follows immediately by inspection, while its security follows from the fact that y_0 and y_1 are computed by independent executions of the Secure_Exp algorithm.
Let us now consider hard commitments. A hard commitment, given the public parameters (p, g, h) and a message m, consists of a pair $\texttt{com} = (g^m \cdot (h^{r_1})^{r_0}, h^{r_1})$ and $\texttt{dec} = (r_0, r_1)$ where r_0 and r_1 are randomly chosen. In this case, clearly the value m need to be private for guaranteeing the hiding property of the commitment. The exponents r_0 and r_1 need to be private since they constitute the decommitment key $\texttt{dec}$. Finally the value h^{r_1} needs to be kept private since it constitutes the second component of $\texttt{com}$. Indeed, if such value becomes public, an attacker that eavesdrops a pair (x, h^{r_1}) knows, w.h.p. that such pair defines a hard commitment, contradicting the indistinguishably of hard and soft commitments. Thus, the Commit procedure can be distributed as follows:

Procedure Commit(p, g, h, m, k^2)

- $r_0, r_1 \in_R Z_p$.
- $w = \text{Secure\_Exp}(g, m, p, k)$
- $y_1 = \text{Secure\_Exp}(h, r_1, p, k)$
- $y_0 = \text{Secure\_Base\_Exp}(y_1, r_0, p, k^2)$
- set $\texttt{com} = (wy_0, y_1)$ and $\texttt{dec} = (r_0, r_1)$

The correctness of the above algorithm can be verified by inspection while, as before, its security follows from the independence of the computations for w, y_1 and y_0.

Distributing the verification. We can now show the distribution of load on the User side. Notice that, the same observations above also apply to the verification procedures described below.

Procedure VerifyOpen$(p, g, h, (C_0, C_1), (r_0, r_1), m, k^2)$

- $\overline{c_1} = \text{Secure\_Exp}(h, r_1, p, k)$
- $\overline{g} = \text{Secure\_Exp}(g, m, p, k)$
- $\overline{c_2} = \text{Secure\_Base\_Exp}(c_1, r_0, p, k^2)$
- if $C_1 = \overline{c_1}$ and $C_0 = \overline{g} \cdot \overline{c_2}$ output "Verified" else output "Failure".

Procedure VerifyTease$(p, g, h, (C_0, C_1), \tau, m, k^2)$

- $\overline{c_1} = \textsc{Secure\_Base\_Exp}(C_1, \tau, p, k^2)$
- $\overline{g} = \textsc{Secure\_Exp}(g, m, p, k)$
- if $C_0 = \overline{g} \cdot \overline{c_1}$ output "Verified" else output "Failure".

4 Certified Information Access via M1-CIA

In this section we describe how to implement the CIA functionality based on M1-CIA. This description resembles the one in [3]. We consider a simple database D associating to a key x a value $D(x) = v$. Let us assume that all keys have the same length ℓ. A reasonable choice is $\ell = 128$ since on one hand it allows to have a large key space and, at the same time, it is possible to use hash functions for reducing the key size to this small value while preserving collision freeness. The database D can thus be represented by a height-ℓ binary tree where leaf numbered x contains the value $v = D(x)$. If no value is associated by the database to key x, the leaf numbered x contains the special value $\perp$.

The CertifiedDBOwner constructs a binary tree that can be described as follows: Leaves of the tree contain the commitment of elements of the database. Each internal node of the tree contains the commitment to the contents of its two children. The commitment to the root of such a tree constitutes the public information that is sent to the PubInfoStorage.

To respond to a query about x, the CertifiedDBOwner simply decommits the corresponding leaf and provides the authenticating path (along with all the decommitments) to the root. The problem with this approach is that it requires time exponential in the height of the tree: if we choose $\ell = 128$, then 2^{128} commitments need to be computed.

This is where M1-CIA helps. Observe that the exponential-size tree has large empty subtrees (that is, subtrees where each leaf is a commitment to $\perp$). Instead of actually computing such a subtree ahead of time, the CertifiedDBOwner forms the root of this subtree as a soft commitment and does not do anything for the rest of the tree. Thus the size of the tree is reduced to at most $2\ell|D|$, where $|D|$ represents the number of element in the database. Responding to a query about x such that $D(x) \neq \perp$ is still done in the same way. If instead $D(x) = \perp$, the CertifiedDBOwner teases the path from the root to x. More precisely, the path from the root to x will consists of hard commitments until the root R of the empty subtree containing x is encountered. All hard commitments from the root of the tree to R are teased to their real values (recall that hard commitments can be teased only to their real value). Then, the CertifiedDBOwner generates a path of soft commitments from R to (the leaf with number) x ending with the commitment of $\perp$. Each soft commitment corresponding to a node along the path is teased to the soft commitments corresponding to its two children. The User simply needs to verify that each teasing has been correctly computed. We stress that for positive queries (that is, queries for x such that $D(x) \neq \perp$) the User expects to see *opening* of hard commitments whereas for negative queries (that is, queries for x such that $D(x) = \perp$) the User expects to see *teasing* of commitments; some of them will be hard commitments and some will be

soft commitments but the user cannot say which ones are which. Due to space limitations, we describe the above procedures in details in Appendix A.

5 The Architectural Design Principles.

In this section we briefly describe the architectural design for a system implementing the primitives described above. We identify four different entities, CertifiedDBOwner, User and PubInfoStorage and Peer.

The applications cooperate as follow. At startup, the PubInfoStorage generates the public parameters that will be used for the M1-CIA implementations. Since public parameters are used both for generating the public information and verifying all the answers to queries, such generation is carried out once, and the parameters are stored in a file.

At this point the CertifiedDBOwner generates the public information with the help of the Peer applications that is run on a set of peers. The public information is sent and stored by the PubInfoStorage. When the User queries the CertifiedDBOwner, he obtains a reply that is verified against the public information held by the PubInfoStorage. We assume point-to-point secure communication among CertifiedDBOwner, User and PubInfoStorage. Furthermore, the communication between any peer and an entity invoking its service is also secured.

In our experience, the most time-consuming operation is the creation of the public information executed by the CertifiedDBOwner. Notice that the distribution of load clearly helps in reducing the time required for such operation if the number t of available peers is bigger than k^2, where k is the security threshold.

A possible way of further reducing the computation time by using pre-computation. We observe that a soft commitment is composed by a pair (g^{r_0}, g^{r_1}) where both r_0 and r_1 are random values. Furthermore, a hard commitment is a pair whose second component is h^r where, again, r is a random value. Clearly it is immaterial whether or not the value r is chosen by the peer. What it does matter is that the CertifiedDBOwner, given the information obtained by the peers, is able to compute some pair $(r', g^{r'})$ (resp., $(r', h^{r'})$) where both components are private.

We can thus introduce a new service, the Batch_Mod_Exp that, given the public parameter held by the PubInfoStorage, computes and stores two lists of pairs, (r, g^r) and $(r', h^{r'})$. Whenever the CertifiedDBOwner needs to compute a soft (resp., hard) commitment, it may simply invoke the Batch_Mod_Exp service that will return the first pair (r, g^r) (resp., (r, h^r)) and remove it from the list.

The pre-computation technique just described can be extremely useful in the case in which the public information associated to the database need to be recomputed frequently. As stated in the introduction, there are no currently available *efficient* solution for implementing a CIA service in case the database is dynamic. Consider, for example, the case in which the database needs to

be modified, say, once per day. The only way of guaranteeing the security of the service is to recompute the public information each time. In this case the company where the CertifiedDBOwner is running, may set up peers on its computers so that they pre-compute the information when they are not used, e.g., at night.

6 Conclusions

In this paper we have presented a distributed architecture for a Certified Information Access system. We have shown that it is possible to securely distribute the load of the most time-consuming operations among a set of untrusted peers. Furthermore, we have presented a solution that allows the usage of pre-computation in order to reduce the time needed by the CertifiedDBOwner for generating the public information.

References

1. Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In Christian Cachin and Jan Camenisch, editors, *International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT 2004)*, volume 3027 of *Lecture Notes in Computer Science*, pages 506–522. Springer, 2004.
2. Dario Catalano, Yevgeniy Dodis, and Ivan Visconti. Mercurial commitments: Minimal assumptions and efficient constructions. In Shai Halevi and Tal Rabin, editors, *Third Theory of Cryptography Conference (TCC 2006)*, volume 3876 of *Lecture Notes in Computer Science*, pages 120–144. Springer, 2006.
3. Melissa Chase, Alexander Healy, Anna Lysyanskaya, Tal Malkin, and Leonid Reyzin. Mercurial commitments with applications to Zero-Knowledge sets. In Ronald Cramer, editor, *24th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT 2005)*, volume 3494 of *Lecture Notes in Computer Science*, pages 422–439. Springer, 2005.
4. Benny Chor, Niv Gilboa, and Moni Naor. Private information retrieval by keywords. In *TR-CS0917, Department of Computer Science, Technion,*, 1997.
5. Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. Protecting data privacy in private information retrieval schemes. *J. Comput. Syst. Sci.*, 60(3):592–629, 2000.
6. Eyal Kushilevitz and Rafail Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *38th Symposium on Foundations of Computer Science (FOCS 1997)*, pages 364–373. IEEE Computer Society, 1997.
7. Moses Liskov. Updatable zero-knowledge databases. In Bimal K. Roy, editor, *11th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT 2005)*, volume 3788 of *Lecture Notes in Computer Science*, pages 174–198. Springer, 2005.
8. Silvio Micali, Michael O. Rabin, and Joe Kilian. Zero-knowledge sets. In *44th Symposium on Foundations of Computer Science (FOCS 2003)*, pages 80–91. IEEE Computer Society, 2003.

9. Sanjeev Kumar Mishra and Palash Sarkar. Symmetrically private information retrieval. In Bimal K. Roy and Eiji Okamoto, editors, *First International Conference in Cryptology in India (Indocrypt 2000)*, volume 1977 of *Lecture Notes in Computer Science*, pages 225–236. Springer, 2000.
10. Wakaha Ogata and Kaoru Kurosawa. Oblivious keyword search. *Journal of Complexity*, 20(2-3):356–371, 2004.
11. Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy*, pages 44–55, 2000.

A Certified Information Access via M1-CIA

In this appendix we describe in more details the generation of the tree of commitments, the construction of a answer by the CertifiedDBOwner and the verification procedure executed by the User.

Generation of Public Information. To generate the public information representing the database D, the CertifiedDBOwner proceeds as follows. We stress, that even though not explicitly specified, all calls to Commit and SoftCommit take as input also the public parameter generated by the PubInfoStorage during the SETUP.

The construction of the tree of commitments starts from its leaves. More precisely, for each x such that $D(x) \neq \perp$, the CertifiedDBOwner produces $(C_x, D_x) = \text{Commit}(D(x))$. Then for each x such that $D(x) = \perp$ but $D(x') \neq \perp$ (where x' is x with the last bit flipped), the CertifiedDBOwner produces $(C_x, D_x) = \text{SoftCommit}$. For all the others x, $C_x = \emptyset$. Now the tree is constructed in a bottom-up fashion as follows: for each level $i = \ell - 1, \cdots, 0$ and for each string s of length i, define C_s as follows:

1. If $C_{s0} \neq \emptyset$ and $C_{s1} \neq \emptyset$, the let $(C_s, D_s) = \text{Commit}((C_{s0}, C_{s1}))$.
2. For all s such that $C_{s'}$ has been defined in the previous step (s' is s with the last bit flipped) but C_s has not, define $(C_s, D_s) = \text{SoftCommit}$.
3. For all other s, define $C_s = \emptyset$.

The value at the root C_ϵ is the public information. If we have $C_\epsilon = \emptyset$ we set $(C_\epsilon, D_\epsilon) = \text{SoftCommit}$.

Answer to a query. This method constructs an object that contains the answer to the query and a proof of validity composed by the decommitments of the corresponding leaf along with the authenticating path (together with all the decommitments) to the root.

More specifically, we distinguish between the case in which the query x is such that $D(x) \neq \perp$ and $D(x) = \perp$. For a string x we denote by $x|_i$ the first i bits of x and by $(x|_i)'$ the first $i-1$ bits of x followed by the i-th bit of x flipped.

If $D(x) \neq \perp$, the authenticating path is computed by sending $D(x)$, the corresponding decommitment key D_x and, for $0 \leq i \leq \ell - 1$, the values $(C_{x|_i 0}, C_{x|_i 1})$ along with the decommitment key $D_{x|_i}$.

Suppose instead that $D(x) = \perp$ and let h be largest value such that $C_{x|_h} \neq \emptyset$, set $(C_x, D_x) = \textsc{Commit}(\perp)$, and build a path from x to $C_{x|_h}$ as follows: set $(C_{x'}, D_{x'}) = \textsc{SoftCommit}$; for each level i from $\ell - 1$ to $h + 1$, define $(C_{x|_i}, D_{x|_i}) = \textsc{Commit}(C_{x|_i 0}, C_{x|_i 1})$, and $(C_{(x|i)'}, D_{(x|i)'}) = \textsc{SoftCommit}$. Note that the only values inside the tree redefined by the above procedure are those that were not defined before.

Let $\tau_x = \textsc{Tease}(D(x), C_x, D_x)$ and $\tau_{x|_i} = \textsc{Tease}((C_{x|_i 0}, C_{x|_i 1}), C_{x|_i}, D_{x|_i})$ for $0 \leq i < \ell$. The response to the query consists of $\perp$ along with its validation path: $(C_{x|_i}, C_{(x|_i)'})$ for $1 \leq i \leq \ell$ and $\tau_{x|_i}$ for $0 \leq i \leq \ell$.

Verification of an Answer. To verify the certified answer, for a query to a key x such that $D(x) \neq \perp$, User executes the VerifyOpen method on all the decommitments received, from the bottom up to the root. The last verification is made against the database commitment that he has previously retrieved from the PubInfoStorage. In case the key that has been queried is not in the database, then the User has to execute VerifyTease instead of VerifyOpen.

Hashing the values. In the discussion above, we have in several points constructed a hard commitment of two hard commitments. This will make the size of the commitment roughly double at each level. Instead we assume that instead of committing to a string (or to a pair of strings) we commit to its hash value computed using a collision-resistant hash function H which hashes down a string to a fixed length ℓ. Notice, again, that $\ell = 128$ is a good choice. Similarly, we can have a database with keys of different length if, instead of storing the pair (x, v) we store the pair $(H(x), v)$.