Oblivious Transfer with Hidden Access Control Policies

Jan Camenisch[1], Maria Dubovitskaya[1], Gregory Neven[1], and Gregory M. Zaverucha[2]

[1] IBM Research - Zurich, Saumerstrasse 4, CH-8803 Ruschlikon Switzerland
[2] Certicom Research, 5520 Explorer Drive Mississauga, ON, L4W 5L1 Canada

Abstract. Consider a database where each record has different access control policies. These policies could be attributes, roles, or rights that the user needs to have in order to access the record. Here we provide a protocol that allows the users to access the database record while: (1) the database does not learn who queries a record; (2) the database does not learn which record is being queried, nor the access control policy of that record; (3) the database does not learn whether a user's attempt to access a record was successful or not; (4) the user can only obtain a single record per query; (5) the user can only access those records for which she has the correct permissions; (6) the user does not learn any other information about the database structure and the access control policies other than whether he was granted access to the queried record, and if so, the content of the record; and (7) the users' credentials can be revoked.

Our scheme builds on the one by Camenisch, Dubovitskaya and Neven (CCS'09), who consider oblivious transfer with access control when the access control policies are public.

Keywords: Privacy, Oblivious Transfer, Anonymous Credentials, Access Control.

1 Introduction

When controlling access to a sensitive resource, it is clear that the applicable access control policies can already reveal too much information about the resource. For example, consider a medical database containing patient records, where the access control policy (ACP) of each record lists the names of the treating doctors. The fact that a patient's record has certain specialists in its ACP leaks information about the patient's disease. Many patients may want to hide, for example, that they are being treated by a plastic surgeon or by a psychiatrist. Also, doctors treating a celebrity may want to remain anonymous to avoid being approached by the press.

As another example, in a multi-user file system, it may be desirable to hide the owner of a file or the groups that have access to it to prevent social engineering attacks, coercion, and bribery. In a military setting, knowing which files are classified "top secret", or even just the percentage of "top secret" files in the system, may help an attacker to focus his attack.

Confidentiality of the stored data and associated ACPs is not the only security concern. Privacy-aware users accessing the database may be worried about malicious database servers prying information from the query traffic. For example, the frequency that a patient's record is accessed gives a good estimate of the seriousness of his

D. Catalano et al. (Eds.): PKC 2011, LNCS 6571, pp. 192–209, 2011.

condition, while the identity of the doctors that access it most frequently may be an indication of the nature of the disorder. Users may therefore prefer to query the database anonymously, i.e., hiding their identity, roles, permissions, etc. from the database server, as well as hiding the index of the queried record. At the same time, the database server wants to rest assured that only permitted users have access to the data, and that they cannot find out who else has access to the data.

1.1 Our Contribution

In this paper we consider access to a database where each record is protected by a (possibly) different access control policy, expressed in terms of the attributes, roles, or rights that a user needs to have to obtain access. To provide the maximal amount of privacy to both users and the database server, we propose a protocol guaranteeing that (1) the database does not learn who queries a record; (2) the database does not learn the index nor the ACP of the queried record; (3) the database does not learn whether a user's attempt to access a record was successful or not; (4) users can only obtain a single record per query, (5) users can only access those records for which they satisfy the ACP; (6) at each query, users learn no more information about the applicable access control policy other than whether they satisfy it or not; and (7) the users' credentials can be revoked. Our ACP structure can be used to implement many practical access control models, including access control matrices, capability lists, role-based access control, and hierarchical access control. This work extends the work of Camenisch, Dubovitskaya, and Neven [8] who consider oblivious transfer with access control when the access control policies are public, i.e., they do not satisfy properties (6) and (7).

1.2 Related Work

Oblivious transfer protocols in their basic form [24,21,11] offer users access to a database without the server learning the contents of the query, but place no restrictions on who can access which records. After Aiello et al. suggested priced oblivious transfer [28], Herranz [18] was the first to add access control restrictions to records, but has users authenticate openly (i.e., non-anonymously) to the server. Later, Coull et al. [9] and Camenisch et al. [8] proposed OT protocols with anonymous access control. In all of these works, however, the access control policies are assumed to be publicly available to all users, and the server notices when a user's attempt to access a record fails.

There is also a line of work devoted to access control with hidden policies and hidden credentials, but none of them consider oblivious access to data, meaning that the server learns which resource is being accessed. In trust negotiation systems [19,26,27], two parties establish trust through iterative disclosure of and requests for credentials. Hidden credentials systems are designed to protect sensitive credentials and policies [4,17]. Neither provide full protection of policies, however, in the sense that the user learns (partial) information about the policy if her credentials satisfy it. The protocol of Frikken et al. [15] does provide full protection, but for arbitrary policies it requires communication exponential in the size of the policies.

Finally, one could always implement a protocol with all desired properties by evaluating an especially designed logical circuit using generic two-party computation

techniques [25], but the cost of this approach would be prohibitive. In particular, the computation and communication cost of each record transfer would be linear in the number of records in the database N, whereas the efficiency of our transfer protocol is independent of N.

2 Definition of OT with Hidden Access Control Policies

An oblivious transfer protocol with hidden access control policies (HAC-OT) is run between an issuer, a database, and one or more users. The issuer provides access credentials to users for the data categories that they are entitled to access. The database hosts a list of records and associates to each record an access control policy (ACP). Users can request individual records from the database, and the request will succeed provided they have the necessary credentials. The ACPs are never revealed.

In a nutshell, a HAC-OT protocol works as follows. The issuer generates its key pair for issuing credentials and publishes the public key as a system-wide parameter. The database server initializes a database containing records protected by access control policies. It generates the encrypted database, which also contains the encrypted access control policies, and makes it available to all users, e.g., by posting it on a website or by distributing it on DVDs. Each user contacts the issuer to obtain a credential that lists all data categories that the user is entitled to access. When she wants to access a record in the database, the user proves to the database in zero-knowledge that her credential contains all the data categories required by the access control policy associated to the record. She performs computations on the encrypted access control rule associated to the desired record so that, with the help of the database, she will obtain the record key if and only if she satisfies the (encrypted) ACP. The database learns nothing about the index of the record that is being accessed, nor about the categories in the access control policy. The database does not even learn whether the user's attempt to obtain a record was successful.

2.1 Setting and Procedures

If $\kappa \in \mathbb{N}$, then 1^κ is the string consisting of κ ones. The empty string is denoted ε. If A is a randomized algorithm, then $y \xleftarrow{\$} \mathsf{A}(x)$ denotes the assignment to y of the output of A on input x when run with fresh random coins.

Unless noted, all algorithms are probabilistic polynomial-time (PPT) and we implicitly assume they take an extra parameter 1^κ in their input, where κ is a security parameter. A function $\nu : \mathbb{N} \to [0,1]$ is *negligible* if for all $c \in \mathbb{N}$ there exists a $\kappa_c \in \mathbb{N}$ such that $\nu(\kappa) < \kappa^{-c}$ for all $\kappa > \kappa_c$.

We consider a limited universe of data categories $\mathcal{C} = \{C_1, \ldots, C_\ell\} \subseteq \{0,1\}^*$. An access control policy $ACP \subseteq \mathcal{C}$ contains those data categories that a user needs to have access to in order to obtain the record. We will usually encode access control policies as vectors $\boldsymbol{c} = (c_1, \ldots, c_\ell) \in \{0,1\}^\ell$, where $c_i = 1$ iff $C_i \in ACP$. A database consists of a list of N pairs $((R_1, ACP_1), \ldots, (R_N, ACP_N))$ of records $R_i \in \{0,1\}^*$ and their associated access control policies $ACP_i \subseteq \mathcal{C}$.

Users hold credentials that certify the list of categories that the user is entitled to access. The list is encoded as a vector $\boldsymbol{d} = (d_1, \ldots, d_\ell) \in \{0,1\}^\ell$, where $d_i = 1$ iff the

user is granted access to category C_i. Letting $\boldsymbol{c} \cdot \boldsymbol{d} = \sum_{i=1}^{\ell} c_i d_i$ and $|\boldsymbol{c}| = \sum_{i=1}^{\ell} c_i$, we say that a user's credential $\boldsymbol{d}$ *covers* an access control policy $\boldsymbol{c}$ iff $\boldsymbol{c} \cdot \boldsymbol{d} = |\boldsymbol{c}|$. This essentially means that users need to have access to *all* categories in the ACP in order to have access to the record. This fits nicely to a number of real-world access control models. For example, to implement role-based access control, where each database record can be accessed by users with one particular role, one sets ℓ to be the number of roles in the system, one sets $c_i = 1$ for the required role i and $c_j = 0$ for all $j \neq i$, and one sets $d_i = 1$ in the users' credentials for all roles i that a user owns. In a hierarchical access control system, users are granted access if their access level is at least that of the resource. For example, in a military context, the levels may be "top secret", "secret", "restricted", and "declassified", so that someone with "top secret" clearance has access to all records. To implement this in our system, one would set ℓ to be the number of levels, set $c_i = 1$ for the level i of the resource, and set $d_j = 1$ for all levels j lower than or equal to i.

Alternatively, one could use a coverage definition where $\boldsymbol{d}$ covers $\boldsymbol{c}$ iff $\boldsymbol{c} \cdot \boldsymbol{d} = |\boldsymbol{d}|$, effectively meaning that all of a user's categories have to appear in the ACP in order to be granted access. Our protocol is easily adapted to implement these semantics. This definition of coverage could be useful to implement simple access control matrices: if ℓ is the number of users, then user i would have a credential with $d_i = 1$, and the ACP sets $c_j = 1$ for all users j that are allowed access.

An oblivious transfer protocol with hidden access control policies (HAC-OT) is six polynomial-time algorithms and protocols, i.e., $\mathcal{HAC\text{-}OT} = (\mathsf{ISetup}, \mathsf{Issue}, \mathsf{Revoke}, \mathsf{DBSetup}, \mathsf{DBVerify}, \mathsf{Transfer})$.

- $\mathsf{ISetup}(\mathcal{C}) \xrightarrow{\$} (pk_{\mathrm{I}}, sk_{\mathrm{I}}, RL)$. The issuer runs the randomized ISetup algorithm to generate a public key pk_{I}, the corresponding secret key sk_{I}, and an initial revocation list RL for security parameter κ and category universe $\mathcal{C}$. The public key and revocation list are published as system-wide parameters.
- Issue: Common input: $pk_{\mathrm{I}}, uid, \boldsymbol{d}$; Issuer input: sk_{I}; User output: $cred_{uid}$ or $\perp$. A user obtains an access credential for a vector of categories $\boldsymbol{d} = (d_1, \ldots, d_\ell) \in \{0,1\}^\ell$ by engaging in the Issue protocol with the issuer. The issuer's public key pk_{I}, the user's identity uid, and the vector $\boldsymbol{d}$ are common inputs. The issuer also uses his secret key sk_{I} as an input. At the end of the protocol, the user obtains the credential $cred_{uid}$.
- $\mathsf{Revoke}(sk_{\mathrm{I}}, RL, uid) \xrightarrow{\$} RL'$. To revoke the credential of user uid, the issuer runs the Revoke algorithm to create an updated revocation list RL' and publishes it as a system-wide parameter.
- $\mathsf{DBSetup}\big(pk_{\mathrm{I}}, DB = (R_i, ACP_i)_{i=1,\ldots,N}\big) \xrightarrow{\$} \big((pk_{\mathrm{DB}}, ER_1, \ldots, ER_N),\ sk_{\mathrm{DB}}\big)$. The database server runs the DBSetup algorithm to create a database containing records $R_1, \ldots, R_N$ protected by access control policies $ACP_1, \ldots, ACP_N$. This algorithm generates the encrypted database consisting of a public key pk_{DB} and the encrypted records along with their encrypted access control policies $ER_1, \ldots, ER_N$. The encrypted database is made available to all users, e.g., by posting it on a website.[1] The database server keeps the secret key sk_{DB} for itself.

[1] We assume that each user obtains a copy of the entire encrypted database. It is impossible to obtain our strong privacy requirements with a single database server without running into either computation or communication complexity that is linear in the database size.

- $\mathsf{DBVerify}(pk_{\mathrm{DB}}, EDB) \rightarrow b$. Upon receiving an encrypted database EDB, all users perform a one-time check to test whether EDB is correctly formed ($b = 1$) or not ($b = 0$).
- Transfer: Common input: $pk_{\mathrm{I}}, RL, pk_{\mathrm{DB}}$; User input: $uid, i, ER_i, cred_{uid}$; Database input: sk_{DB}; User output: R_i or $\perp$. When the user wants to access a record in the database, she engages in a Transfer protocol with the database server. Common inputs are the issuer's public key pk_{I}, the revocation list RL, and the database's public key pk_{DB}. The user has as a secret input her identity uid, her selection index $i \in \{1, \ldots, N\}$, the encrypted record with encrypted ACP, and her credential $cred_{uid}$. The database server uses its secret key sk_{DB} as a private input. At the end of the protocol, the user obtains the database record R_i if her credential satisfies the ACP, or receives $\perp$ if not.

We assume that all communication links are private. We also assume that the communication links between a user and the issuer are authenticated, so that the issuer always knows to which user it is issuing a credential. The communication links between a user and the database are assumed to be anonymous, so that the database does not know which user is making a record query. (Authenticated communication channels between users and the database would obviously ruin the strong anonymity properties of our protocol).

2.2 Security Definitions

We define security of an HAC-OT protocol through indistinguishability of a real-world and an ideal-world experiment as introduced by the UC framework [5,6] and the reactive systems security models [22,23]. The definitions we give, however, do not entail all formalities necessary to fit one of these frameworks; our goal here is solely to prove our scheme secure.

We summarize the ideas underlying these models. In the real world there are a number of players, who run some cryptographic protocols with each other, an adversary A, who controls some of the players, and an environment $\mathcal{E}$. The environment provides the inputs to the honest players and receives their outputs and interacts arbitrarily with the adversary. The dishonest players are subsumed into the adversary.

In the ideal world, we have the same players. However, they do not run any cryptographic protocols but send all their inputs to and receive all their outputs from an ideal all-trusted party T. This party computes the output of the players from their inputs, i.e., applies the functionality that the cryptographic protocol(s) are supposed to realize. The environment again provides the inputs to and receives the output from the honest players, and interacts arbitrarily with the adversary controlling the dishonest players. A set of cryptographic protocols is said to securely implement a functionality if for every real-world adversary A and every environment $\mathcal{E}$ there exists an ideal-world simulator A′ controlling the same parties in the ideal world as A does in the real world, such that the environment cannot distinguish whether it is run in the real world interacting with A, or whether it is run in the ideal world interacting with the simulator A′.

Definition 1. *Let* $\mathbf{Real}_{\mathcal{E},\mathsf{A}}(\kappa)$ *denote the probability that* $\mathcal{E}$ *outputs 1 when run in the real world with* A *and let* $\mathbf{Ideal}_{\mathcal{E},\mathsf{A}'}(\kappa)$ *denote the probability that* $\mathcal{E}$ *outputs 1 when*

run in the ideal world with A'*, then the set of cryptographic protocols is said to securely implement functionality* T *if* $\mathbf{Real}_{\mathcal{E},\mathsf{A}}(\kappa) - \mathbf{Ideal}_{\mathcal{E},\mathsf{A}'}(\kappa)$ *is a negligible function in* κ*.*

THE REAL WORLD. We first describe how the real-world algorithms presented in §2.1 are orchestrated when all participants are honest, i.e., honest real-world users $\mathsf{U}_1, \ldots,$ U_M, an honest issuer I, and an honest database DB. If parties are controlled by the real-world adversary A, they can arbitrarily deviate from the behavior described below.

All begins with I generating a key pair and an initial revocation list $(pk_{\mathrm{I}}, sk_{\mathrm{I}}, RL) \xleftarrow{\$} \mathsf{ISetup}(\mathcal{C})$ and sending (pk_{I}, RL) to all users $\mathsf{U}_1, \ldots, \mathsf{U}_M$ and the database DB.

When the environment $\mathcal{E}$ sends a message $(\texttt{initdb}, DB = (R_i, ACP_i)_{i=1,\ldots,N})$ to the database DB, the latter encrypts DB by running $(EDB, sk_{\mathrm{DB}}) \xleftarrow{\$} \mathsf{DBSetup}(pk_{\mathrm{I}}, DB)$, and sends the encrypted database $EDB = (pk_{\mathrm{DB}}, ER_1, \ldots ER_N)$ to all users $\mathsf{U}_1, \ldots, \mathsf{U}_M$. All users execute a $\mathsf{DBVerify}$ protocol with the database and, if it returns 1, return a message $(\texttt{initdb}, N)$ to the environment.

When $\mathcal{E}$ sends a message $(\texttt{issue}, \boldsymbol{d})$ to user U_{uid}, she engages in an Issue protocol with I on common input pk_{I}, uid, and a vector $\boldsymbol{d}$ indicating which categories the user is allowed to access, with I using sk_{I} as its secret input. Eventually, U_{uid} obtains the access credential $cred_{uid}$. User U_{uid} returns a message $(\texttt{issue}, b)$ to the environment indicating whether the issue protocol succeeded ($b = 1$) or failed ($b = 0$). Each user will engage in an Issue protocol only once; if she receives a second message $(\texttt{issue}, \boldsymbol{d})$ from the environment, she simply returns $(\texttt{issue}, 0)$.

When $\mathcal{E}$ sends a message $(\texttt{revoke}, uid)$ to the issuer I, it creates a new revocation list $RL' \xleftarrow{\$} \texttt{revoke}(sk_{\mathrm{I}}, RL, uid)$ based on the old revocation list RL and the user identity uid to be revoked. The issuer returns $(\texttt{revoke}, uid)$ to the environment.

When $\mathcal{E}$ sends a message $(\texttt{transfer}, i)$ to user U_{uid}, then U_{uid} engages in a $\mathsf{Transfer}$ protocol with DB on common input pk_{I} and pk_{DB}, on U_{uid}'s private input i and her credential $cred_{uid}$, and on DB's private input sk_{DB}. As a result of the protocol U_{uid} obtains the record R_i, or $\perp$ indicating failure. If the transfer succeeded the user returns $(\texttt{transfer}, i, R_i)$ to the environment; if it failed she returns $(\texttt{transfer}, i, \perp)$. We note that DB does not return any outputs to the environment.

THE IDEAL WORLD. In the ideal world, all participants communicate through a trusted party T which implements the functionality of our protocol. We describe the behavior of T on the inputs of the ideal-world users $\mathsf{U}'_1, \ldots, \mathsf{U}'_M$, the ideal-world issuer I', and the ideal-world database DB'.

The trusted party T maintains an initially empty vector $\boldsymbol{\varepsilon}$ for each user U'_{uid}, an initially empty list of revoked users RL, and sets $DB \leftarrow \perp$. It responds to queries from the different parties as follows.

- Upon receiving $(\texttt{initdb}, N)$ from DB', T sets $DB \leftarrow (\varepsilon_i, \varepsilon_i)_{i=1,\ldots,N}$ and sends the message $(\texttt{initdb}, N)$ to all users. All users send the message $(\texttt{initdb}, N)$ to the environment.
- Upon receiving $(\texttt{issue}, \boldsymbol{d})$ from U'_{uid} for the first time, T sends $(\texttt{issue}, \mathsf{U}'_{uid}, \boldsymbol{d})$ to I' who sends back a bit b. If $b = 1$ then T initializes the category vector $\boldsymbol{d}_{uid} \leftarrow \boldsymbol{d}$ for the user U'_{uid} and sends $(\texttt{issue}, 1)$ to U'_{uid}; otherwise it simply sends $(\texttt{issue}, 0)$ to U'_{uid}. T responds to all subsequent messages $(\texttt{issue}, \boldsymbol{d})$ from the same user U'_{uid} with $(\texttt{issue}, 0)$.

- Upon receiving $(\texttt{revoke}, uid)$ from I', T adds U'_{uid} to the list of revoked users by setting $RL \leftarrow RL \cup \{uid\}$.
- Upon receiving $(\texttt{transfer}, i)$ from U'_{uid}, T proceeds as follows. If $DB = \perp$, it sends $(\texttt{transfer}, \perp)$ back to U'_{uid}. If $DB = (\varepsilon_i, \varepsilon_i)_{i=1,\ldots,N}$ it sends $(\texttt{transfer}, \varepsilon)$ to DB' who sends back a bit b. If $b = 1$ it also sends the whole database $DB = (R_i, ACP_i)_{i=1,\ldots,N}$, and the T sets $DB = (R_i, ACP_i)_{i=1,\ldots,N}$. If the database DB already contained records, T sends $\texttt{transfer}$ to DB', who sends back just a bit b. If $b = 1$, $\boldsymbol{d}_{uid}$ covers ACP_i, and $uid \notin RL$, then it sends $(\texttt{transfer}, R_i)$ to U'_{uid}; otherwise, it sends $(\texttt{transfer}, \perp)$ to U'_{uid}.

The ideal-world parties $\mathsf{U}'_1, \ldots, \mathsf{U}'_M, \mathsf{I}', \mathsf{DB}'$ simply relay inputs and outputs between the environment $\mathcal{E}$ and the trusted party T.

SECURITY PROPERTIES. It is easy to see that the ideal world definition implies that the users' privacy is protected:

An adversary, controlling all parties except some honest users, cannot tell which of the users access which record nor whether the attempt was successful.

Also, the database is guaranteed that (potentially malicious) users can only access the records for which they were issued credentials and that users do not learn any information about the access control lists apart from the fact whether or not their credentials allow them to access a record. We note that colluding users cannot pool their credentials nor can they obtain access with revoked credentials.

3 Randomizing and Extending Groth-Sahai Proofs

Let $\mathsf{Pg}(1^\kappa)$ be a pairing group generator that on input 1^κ outputs descriptions of multiplicative groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathrm{T}$ of prime order p where $|p| > \kappa$. Let $\mathsf{Pg}(p)$ be a pairing group generator that on input a prime p outputs descriptions of multiplicative groups $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_\mathrm{T}$ of order p. Let $\mathbb{G}_1^* = \mathbb{G}_1 \setminus \{1\}, \mathbb{G}_2^* = \mathbb{G}_2 \setminus \{1\}$ and let $g_1 \in \mathbb{G}_1^*, g_2 \in \mathbb{G}_2^*$. The generated groups are such that there exists an admissible bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_\mathrm{T}$, meaning that (1) for all $a, b \in \mathbb{Z}_p$ it holds that $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$; (2) $e(g_1, g_2) \neq 1$; and (3) the bilinear map is efficiently computable. The group setting $\mathit{GroupSet}$ is a tuple $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, e)$.

Groth and Sahai [16] present non-interactive witness-indistinguishable proofs of knowledge for three types of equations involving bilinear groups. These are: (i) pairing product equations, (ii) multi-exponentiation equations, and (iii) quadratic equations modulo the group order. Our protocol primarily uses proofs of the second type of equation, which can be made zero knowledge (ZK).

Belenkiy et al. [3] show that the Groth-Sahai proofs can be randomized such that they still prove the same statement but different proofs for the same statement are indistinguishable. In this paper, we extend these ideas: we take a proof for one statement and transform and randomize it into a proof of a related statement.

Independently of our work, Dodis et al. [12] also suggest to use GS-proofs of an old statement to construct a GS-proof for a modified statement. However, their approach only exploits the additive homomorphism of the GS-proof scheme, i.e., it allows one to modify an old witness by adding a new value to it and construct a proof for the new

witness, without knowledge of the old one. We suggest a modification scheme for the GS proof that enables modifying the witness in a more general way using multiplication and addition.

We describe how this is done in the following. We start with the descriptions of the basic algorithms of an instantiation of the Groth-Sahai proof system for multi-exponentiation equations for the prime-order groups in the group setting $GroupSet = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, e)$, i.e., the proof system

$$\mathsf{NIZKP}_{\mathsf{GS}}\{((x_{ij})_{i=1,\ldots,M,\ j=1,\ldots,\ell}) \ : \ \bigwedge_{i=1}^{M} y_i = \prod_{j=1}^{\ell} g_j^{x_{ij}}\},$$

where the y_i's and g_j's are public group elements of $\mathbb{G}_1$ (cf. [7]). In the following let $stmt = ((x_{ij})_{j=1,\ldots,\ell;i=1,\ldots,M}) : \bigwedge_{i=1}^{M} y_i = \prod_{j=1}^{\ell} g_j^{x_{ij}}$. The proof system for $GroupSet$ consists of three algorithms $\mathsf{GSSetup}$, $\mathsf{GSProve}$, and $\mathsf{GSVerify}$. A trusted third party generates the common (public) reference string by running $CRS \leftarrow \mathsf{GSSetup}(GroupSet)$. A prover generates a proof as $\pi \leftarrow \mathsf{GSProve}(CRS, stmt, (y_i), (g_j), (x_{ij}))$ and a verifier checks it via $b \leftarrow \mathsf{GSVerify}(CRS, \pi, stmt, (y_i), (g_j))$, where $b = 1$ if π is true w.r.t. $stmt$ and $b = 0$ otherwise. We now present these algorithms in detail, based on the XDDH assumption [16,7]. (For ease of notation, we will denote by (y_i), (g_j), and (x_{ij}) the lists $(y_1, \ldots y_M)$, $(g_1, \ldots, g_\ell)$, and $(x_{11}, \ldots, x_{M\ell})$ whenever the indices are clear from the context).

$\mathsf{GSSetup}(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, e) \xrightarrow{\$} CRS$: Return $CRS = (\chi_1, \chi_2, \gamma_1, \gamma_2) \xleftarrow{\$} \mathbb{G}_2^4$.

$\mathsf{GSProve}(CRS, stmt, (y_i), (g_j), (x_{ij})) \xrightarrow{\$} \pi$:

1. Pick $r_{ij} \xleftarrow{\$} \mathbb{Z}_p$ for $i = 1 \ldots M$ and $j = 1, \ldots, \ell$.
2. For each x_{ij} in (x_{ij}) compute the set of commitments $C_{ij}^{(1)} \leftarrow \gamma_1^{x_{ij}} \chi_1^{r_{ij}}$; $C_{ij}^{(2)} \leftarrow \gamma_2^{x_{ij}} \chi_2^{r_{ij}}$.
3. For each y_i in (y_i) compute $p_i = \prod_{j=1}^{\ell} g_j^{r_{ij}}$.
4. Return $\pi \leftarrow (p_i, (C_{ij}^{(1)}, C_{ij}^{(2)})_{j=1,\ldots\ell})_{i=1\ldots M}$.

$\mathsf{GSVerify}(CRS, \pi, stmt, (y_j), (g_i)) \xrightarrow{\$} b$:

1. If for all $i = 1 \ldots M$ we have $(\prod_{j=1}^{\ell} e(g_j, C_{ij}^{(1)}) = e(y_i, \gamma_1) e(p_i, \chi_1)) \wedge (\prod_{j=1}^{\ell} e(g_j, C_{ij}^{(2)}) = e(y_i, \gamma_2) e(p_i, \chi_2))$ then return $b \leftarrow 1$, else return $b \leftarrow 0$.

For the security properties of these algorithms we refer to Groth and Sahai [16] and Camenisch et al. [7].

Now, like Belenkiy et al. [3], we extend this basic system with a fourth algorithm GSRand which allows anyone to take a proof π and randomize it to obtain a proof π' for the same statement without knowledge of the witnesses (x_{ij}). Still, the proofs π and π' have the same distribution. This algorithm is as follows.

$\mathsf{GSRand}(CRS, \pi, stmt, (y_i), (g_j)) \xrightarrow{\$} \pi'$:

1. If $0 = \mathsf{GSVerify}(CRS, \pi, stmt, (y_i), (g_j))$ abort.
2. Pick $r'_{ij} \xleftarrow{\$} \mathbb{Z}_p$ for $i = 1 \ldots M$ and $j = 1, \ldots, \ell$.

3. Re-randomize all commitments:
   For every x_{ij} compute $C'^{(1)}_{ij} = C^{(1)}_{ij}\chi_1^{r'_{ij}}; C'^{(2)}_{ij} = C^{(2)}_{ij}\chi_2^{r'_{ij}}$.
4. Re-randomize p_i (consistent with the new randomness), by computing
   $p_i = p_i \prod_{j=1}^{\ell} g_j^{r'_{ij}}$.
5. Return $\pi' \leftarrow (p'_i, (C'^{(1)}_{ij}, C'^{(2)}_{ij})_{j=1,\ldots\ell})_{i=1\ldots M}$.

It is not hard to see that the proof π' has the same distribution as π and will be accepted by GSVerify. Also note that all the security properties of the proof system are retained (the algorithm essentially only randomizes, cf. [7]).

We now extend the above ideas to a fifth, and new algorithm GSRMod, which allows us not only to re-randomize the proof π for the statement $stmt$ but also to extend it to a proof $\hat{\pi}$ for the related statement

$$stmt' = ((\hat{x}_{ij})_{j=1,\ldots,\ell;i=1,\ldots,M}) : \bigwedge_{i=1}^{M} \hat{y}_i = \prod_{j=1}^{\ell} g_j^{\hat{x}_{ij}}$$

where $\hat{y}_i = \prod_{j=1}^{M} y_j^{x'_j} \prod_{j=1}^{\ell} g_j^{x'_{ij}}$. Similarly to just randomizing a proof, it is sufficient to know x'_j and x'_{ij} to do this proof modification, i.e., knowledge of $\hat{x}_{ij}$ is not required. Note that $\hat{x}_{ij} = x'_{ij} + \sum_{k=1}^{M} x_{kj} \cdot x'_k$ will hold w.r.t. the original witnesses x_{ij}.

$\mathsf{GSRMod}(CRS, \pi, stmt', stmt, (\hat{y}_i), (y_i), (g_j), (x'_k)(x'_{ij})) \xrightarrow{\$} \pi'$:

1. If $0 = \mathsf{GSVerify}(CRS, \pi, stmt, (y_i), (g_j))$ abort.
2. Pick $r'_{ij} \xleftarrow{\$} \mathbb{Z}_p$ for $i = 1, \ldots, M$ and $j = 1, \ldots, \ell$.
3. Create commitments for each $\hat{x}_{ij}$ using old commitments:
   $\hat{C}^{(1)}_{ij} = \prod_{k=1}^{M} (C^{(1)}_{kj})^{x'_k} \gamma_1^{x'_{ij}} \chi_1^{r'_{ij}}$; $\hat{C}^{(2)}_{ij} = \prod_{k=1}^{M} (C^{(2)}_{kj})^{x'_k} \gamma_2^{x'_{ij}} \chi_2^{r'_{ij}}$.
4. Re-randomize and modify p_i (consistent with the new witnesses and randomness), by computing
   $\hat{p}_i = \prod_{j=1}^{M} (p_j)^{x'_j} \prod_{j=1}^{\ell} g_j^{r'_{ij}}$.
5. Return $\hat{\pi} \leftarrow (\hat{p}_i, (\hat{C}^{(1)}_{ij}, \hat{C}^{(2)}_{ij})_{j=1,\ldots\ell})_{i=1\ldots M}$.

Again, it is not hard to see that all security properties are retained (cf. [7]).

4 Our Construction

The main ideas underlying our protocol are as follows: the database server starts by encrypting each record with a key that is at the same time a signature on the index of the record and on the access control policy for the record. Furthermore, the server ElGamal-encrypts the access control policy for each record and creates a commitment to the policy. It then provides a non-interactive GS proof that the commitment and the encryptions are consistent. All of these values are then published as the encrypted database.

To be able to access the records, users are issued credentials for the list of data categories they are allowed to access. To revoke a user's credential, we use ideas from

the revocation scheme by Nakanishi et al. [20] about revocable group signatures, where the revoked user identities are sorted in lexicographical order and the revocation list contains signatures on each pair of neighboring revoked user identities. To prove that her credential is not revoked, the user needs to show that her identity lies within the open interval defined by one of the signed identity pairs in the current revocation list. For this purpose the issuer signs all possible "distances" within such intervals, called "revocation distances". The user then proves that she possesses a valid signed pair of revoked user identities and valid signatures on the distances to the edges of the revocation interval. We note that to improve efficiency, one could make the maximal interval sizes smaller by revoking a number of dummy user identities by default.

When the user wants to access a record i, she re-encrypts the encrypted ACP for that record under her own freshly generated public key. She also randomizes the commitment to the ACP and then modifies the original GS proof into a new one proving that the new encryptions and the new commitment are consistent. The user also blinds the database server's signature on the ACP. Using the homomorphism of ElGamal encryption, the user computes an encryption of $\delta = \sum c_{ij} d_j - \sum c_{ij}$. Note that δ is 0 if the user is allowed access and non-zero otherwise. Finally, the user sends these values to the database server and proves in zero-knowledge that 1) she computed the encryption of δ correctly from the modified encryptions and w.r.t. the d_j that appear in her credential, that 2) the blinded signature is a valid signature by the database on the ACP values in the randomized commitment (without knowing these values of course), and that 3) her credential was not revoked.

If all of these proofs verify correctly, the database server uses the blinded signature to compute the blinded key of the record and "folds it into" the encryption of δ so that it contains the blinded key if $\delta = 0$ and contains a random plaintext otherwise. The server sends these values to the user and proves that they were computed correctly. Upon receipt, the user decrypts and unblinds the record key, and decrypts the record.

4.1 Issuer Setup

We now describe each step of our scheme in detail. We begin with the setup procedures of the issuer and the database provider. Users do not have their own setup procedure.

To set up its keys, the issuer runs the randomized ISetup algorithm displayed in Figure 1. This will generate groups of prime order p, a public key pk_{I} and corresponding secret key sk_{I} for security parameter κ and category universe $\mathcal{C}$.

The values $g_I, y_I, h_0, \ldots, h_{\ell+1}, u, w$ from the issuer's public key and the corresponding x_I from the secret key are used to issue credentials. The values $\hat{g}, \hat{g}_1, \hat{g}_3, \hat{g}_4, y_r$

$(\overline{\mathbb{G}}_1, \overline{\mathbb{G}}_2, \overline{\mathbb{G}}_{\mathrm{T}}, p) \xleftarrow{\$} \mathsf{Pg}(1^\kappa)$; $g_I, h_0, \ldots, h_{\ell+1}, u, w, \hat{g}, \hat{g}_1, \hat{g}_2, \hat{g}_3, \hat{g}_4 \xleftarrow{\$} \overline{\mathbb{G}}_1$;
$h_I, h_r, h_\Delta \xleftarrow{\$} \overline{\mathbb{G}}_2$; $g_t, h_t \xleftarrow{\$} \overline{\mathbb{G}}_{\mathrm{T}}$; $x_I, x_\Delta, x_r \xleftarrow{\$} \mathbb{Z}_p$; $y_I \leftarrow h_I^{x_I}$; $y_\Delta \leftarrow h_\Delta^{x_\Delta}$; $y_r \leftarrow h_r^{x_r}$;
$r, s \xleftarrow{\$} \mathbb{Z}_p$; $S \leftarrow (\hat{g}\hat{g}_1^1 \hat{g}_2^0 \hat{g}_3^{\Delta_{\max}+1} \hat{g}_4^r)^{1/(x_r+s)}$.
For $i = 1, \ldots, \Delta_{\max}$ do $y_\Delta^{(i)} \leftarrow g_I^{1/(x_\Delta+i)}$.
Return $(sk_{\mathrm{I}} = (x_I, x_\Delta, x_r), pk_{\mathrm{I}} = (g_I, h_0, \ldots, h_{\ell+1}, u, w, \hat{g}, \hat{g}_1, \hat{g}_3, \hat{g}_4, h_I, h_\Delta, h_r, g_t, h_t, y_I,$
$y_r, y_\Delta, y_\Delta^{(1)}, \ldots, y_\Delta^{(\Delta_{\max})}), RL = (1, \{0, \Delta_{\max}+1\}, \{(0, \Delta_{\max}+1, S, r, s)\}))$.

Fig. 1. Issuer Setup algorithm $\mathsf{ISetup}(\mathcal{C})$

and x_r are used to sign pairs of the revoked user identities in the revocation list. And, finally, the issuer uses the Boneh-Boyen signature scheme with secret key x_Δ and public key y_Δ to sign all possible "revocation distances" and makes the set of these signatures $(y_\Delta^{(i)})_{i=1}^{\Delta_{\max}}$ a part of its public key. It will later become apparent what these revocation distances are (users will need them for proving that her credential is not on the list of revoked credentials).

He publishes the public key as a system-wide parameter.

4.2 Issuing Credentials

To be able to make database queries, a user needs to obtain a credential for the categories that she is allowed to access. To this end, the user runs the Issue protocol with the issuer as depicted in Figure 2. How the issuer determines which user has access to which categories is of course out of scope of the scheme.

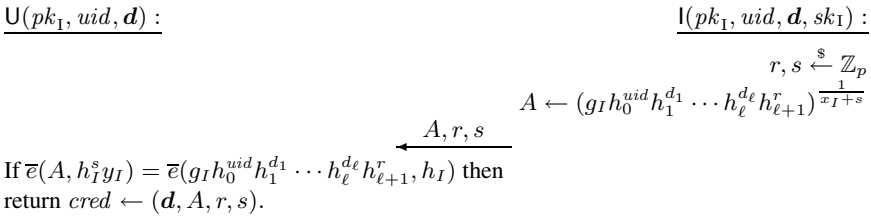

Fig. 2. Issue protocol Issue()

As a result of the issuing protocol, the user will obtain an access credential for the vector $\boldsymbol{d}$. This credential is a triple (A, r, s), which is a signature on the user's identity uid and the messages $\boldsymbol{d}$, using the signature scheme proposed and proved secure by Au et al. [1]. It is based on the schemes of Camenisch and Lysyanskaya [10] and of Boneh et al. [2].

4.3 Revoking Credentials

To revoke a user uid's credential, the issuer runs the revocation algorithm Revoke$(pk_\mathrm{I}, sk_\mathrm{I}, RL, uid)$. Recall that in our model each user has only one credential, in which the user's identity uid is embedded. We implement this revocation list with the ideas of Nakanishi et al. [20] for revocation in the context of group signatures. Thus the revocation list $RL = (t, R_1, R_2)$ consists of its current version number t, a set R_1 containing the identities of all revoked users, and a set R_2 of signatures on each pair of neighboring uid's in R_1 (according to their lexicographic order). When the issuer revokes a credential, it takes the latest revocation list $RL = (t, R_1, R_2)$ and constructs the updated revocation list by setting the version number to $t' = t + 1$, by adding the revoked user identity uid to R_1, by sorting this list lexicographically, and by issuing

Parse RL as (t, R_1, R_2) ; $t' \leftarrow t+1$; $R_1' \leftarrow R_1 \cup \{uid\}$; $R_2' \leftarrow \emptyset$.
Sort R_1' lexicographically as $(uid_1, \ldots, uid_n)$.
For $i = 1, \ldots, n-1$ do
$\hat{r}, \hat{s} \xleftarrow{\$} \mathbb{Z}_p$; $S \leftarrow (\hat{g}\hat{g}_1^{t'}\hat{g}_2^{uid_i}\hat{g}_3^{uid_{i+1}}\hat{g}_4^{\hat{r}})^{1/(x_r+\hat{s})}$; $R_2' \leftarrow R_2' \cup \{(t', uid_i, uid_{i+1}, S, \hat{r}, \hat{s})\}$.
Return $RL' = (t', R_1', R_2')$.

Fig. 3. Revocation algorithm $\mathsf{Revoke}(pk_{\mathrm{I}}, sk_{\mathrm{I}}, RL, uid)$

a new set R_2' of signatures on each pair of neighboring revoked user identities in the updated set R_1'. This algorithm is described in detail in Figure 3.

4.4 Database Setup

To set up the database, the database server runs the algorithm shown in Figure 4. That is, it uses the issuer's public key and a pairing group generator to create groups of the same order p and generate keys for encrypting records. First the database provider generates its public and private keys to encrypt records.

Then the database creates a signature σ_i to bind the ACP and index to the encrypted record and "randomizes" it with value v_i. It also computes a commitment V_i to the index, ACP and v_i. The commitment V_i will be used for signature verification.

Next it encrypts each record R_i as (σ_i, F_i), each with its own key σ_i. In fact, these keys are verifiably pseudo random values [13], but are at the same time signatures under the the database provider's secret key (x_{DB}) on the index of the record (i) and the

1. Generate system parameters and keys
   $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T) \xleftarrow{\$} \mathsf{Pg}(p)$; $g, h, h_{DB} \xleftarrow{\$} \mathbb{G}_1^*$; $g' \xleftarrow{\$} \mathbb{G}_2^*$; $H \leftarrow e(h_{DB}, g')$;
   $x_e, x_{\mathrm{DB}} \xleftarrow{\$} \mathbb{Z}_p$; $y_e \leftarrow g^{x_e}$; $y_{\mathrm{DB}} \leftarrow g^{x_{\mathrm{DB}}}$; $CRS \leftarrow \mathsf{GSSetup}(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, e)$;
   For $i = 1, \ldots, \ell+1$ do $\{x_i \xleftarrow{\$} \mathbb{Z}_p$; $y_i \leftarrow g^{x_i}\}$;
   $sk_{\mathrm{DB}} \leftarrow (h_{DB}, x_e, x_{\mathrm{DB}}, x_1, \ldots, x_{\ell+1})$; $pk_{\mathrm{DB}} \leftarrow (g, g', H, h, y_e, y_{\mathrm{DB}}, y_1, \ldots, y_{\ell+1})$.
2. Create an encrypted database
   For $i = 1, \ldots, N$ do
   2.1 Parse ACP vector $\boldsymbol{c_i}$ as $(c_{i1}, \ldots c_{i\ell})$;
   2.2 Sign and encrypt records:
   $v_i \xleftarrow{\$} \mathbb{Z}_p$; $V_i \leftarrow g^i y_1^{c_{i1}} \ldots y_\ell^{c_{i\ell}} y_{\ell+1}^{v_i}$; $\sigma_i \leftarrow g'^{\frac{1}{x_{\mathrm{DB}}+i+\sum_{j=1}^{\ell} x_j \cdot c_{ij} + x_{\ell+1} v_i}}$;
   $F_i \leftarrow e(h_{DB}, \sigma_i) \cdot R_i$.
   2.3 Encrypt categories from the record's ACP:
   For each bit c_{ij} generate $r_{ij} \xleftarrow{\$} \mathbb{Z}_p, j = 1 \ldots \ell$;
   $E_{ij}^{(1)} = g^{c_{ij}} y_e^{r_{ij}}$; $E_{ij}^{(2)} = g^{r_{ij}}$; $E_{ij} = (E_{ij}^{(1)}, E_{ij}^{(2)})$
   2.4 Generate GS proof that all keys, signatures and encryptions were computed correctly
   $\pi_i = \mathsf{GSProve}\big(CRS, stmt_i, ((E_{ij}), V_i), (g, y_e, (y_j)), (v_i, (c_{ij}), (r_{ij}))\big)$
   2.5 $ER_i \leftarrow (\sigma_i, V_i, F_i, (E_{i1}, \ldots, E_{i\ell}), \pi_i)$
3. Publish an encrypted database and public key
   Return $EDB \leftarrow \big((pk_{\mathrm{DB}}, ER_1, \ldots, ER_N),\ sk_{\mathrm{DB}}\big)$

Fig. 4. Database Setup algorithm $\mathsf{DBSetup}\big(pk_{\mathrm{I}}, DB = (R_i, \boldsymbol{c_i})_{i=1,\ldots,N}\big)$

categories defined in the access control policy for the record ($\boldsymbol{c}_i$). The pairs (σ_i, F_i) can be seen as an ElGamal encryption [14] in $\mathbb{G}_\mathrm{T}$ of R_i under the public key H. During the transfer phase, this verifiability allows the database to check that the user is requesting the decryption key for a record with an access control policy satisfied by the user's credential.

To hide the records' ACPs, the database generates ElGamal encryptions of each bit c_{ij} and provides a NIZK GS-proof π_i to prove knowledge of the plaintexts. The statement for this proof for record i is

$$stmt_i = (v_i, c_{ij}, r_{ij}) : V_i = g^i y_1^{c_{i1}} \dots y_l^{c_{il}} y_{l+1}^{v_i} \bigwedge_{j=1}^{\ell} \left(E_{ij}^{(1)} = g^{c_{ij}} y_e^{r_{ij}} \wedge E_{ij}^{(2)} = g_e^{r_{ij}}\right).$$

4.5 Accessing a Record

After having obtained the encrypted database, the user verifies it for correctness by running for each record R_i the step, denoted as $\mathsf{DBVerify}(pk_\mathrm{DB}, EDB)$ and defined as: $\mathsf{GSVerify}(CRS, \pi_i, stmt_i, ((E_{ij}), V_i), (g, y_e, (y_j))) \wedge (e(y_{DB} V_i, \sigma_i) \stackrel{?}{=} e(g, g'))$.

When the user wants to access a record in the database, she engages in a Transfer protocol (Figure 5) with the database server.

The input of the database server is its secret and public key as well as the public key of the issuer. The input of the user is the public keys of the issuer and the database, the index i of the record she wants to access, her credential, and the encrypted record $ER_i = ((\sigma_i, V_i, F_i, (E_{i1}, \dots, E_{i\ell}), \pi_i))$.

At a high level, the protocol has three main steps. First, the user takes the encrypted ACP for the record she wants to access and adds a second layer of encryption to it, using a freshly generated key pair (x_u, y_u). Using the homomorphic properties of the encryption scheme, the user's categories in her credential and ACP for the record are compared by constructing a ciphertext $(D_i^{(1)}, D_i^{(2)})$ that encrypts zero if the user's credential satisfies the ACP, and a non-zero value if it does not. Then, the resulting ciphertext is sent to the database together with a proof PK_1 by the user that she constructed it correctly w.r.t. to the credential she possesses and the encrypted database. If that proof is valid, the database removes one layer of encryption and returns the result to the user. Finally, the user removes the remaining layer of encryption to recover the key for the record (or a random value if access is not granted).

Now we describe each step in greater detail. The user takes the ElGamal encryptions of each category bit for the record she wants to access and re-encrypts them with her own key. Then, using these values, she calculates an ElGamal encryption of $\delta = (\sum_{j=1}^{\ell} c_{ij} d_j - \sum_{j=1}^{\ell} c_{ij})$, which will be 0 if and only if $c_{ij} = d_j$. She then re-randomizes and modifies the GS proof π into the new one π' for the statement

$$stmt'_i = (v_i, i, (c_{ij}), (r_{ij}), (r'_{ij}), x_V) : V'_i = g^i y_1^{c_{i1}} \dots y_\ell^{c_{i\ell}} y_{\ell+1}^{v_i} h^{x_V} \bigwedge_{j=1}^{\ell} \left(E_{ij}^{\prime(1)} = g^{c_{ij}} (y_e y_u)^{r_{ij}+r'_{ij}} \wedge E_{ij}^{\prime(2)} = g_e^{r_{ij}+r'_{ij}}\right)$$

to prove that the new encryptions are consistent with the new commitment V'_i.

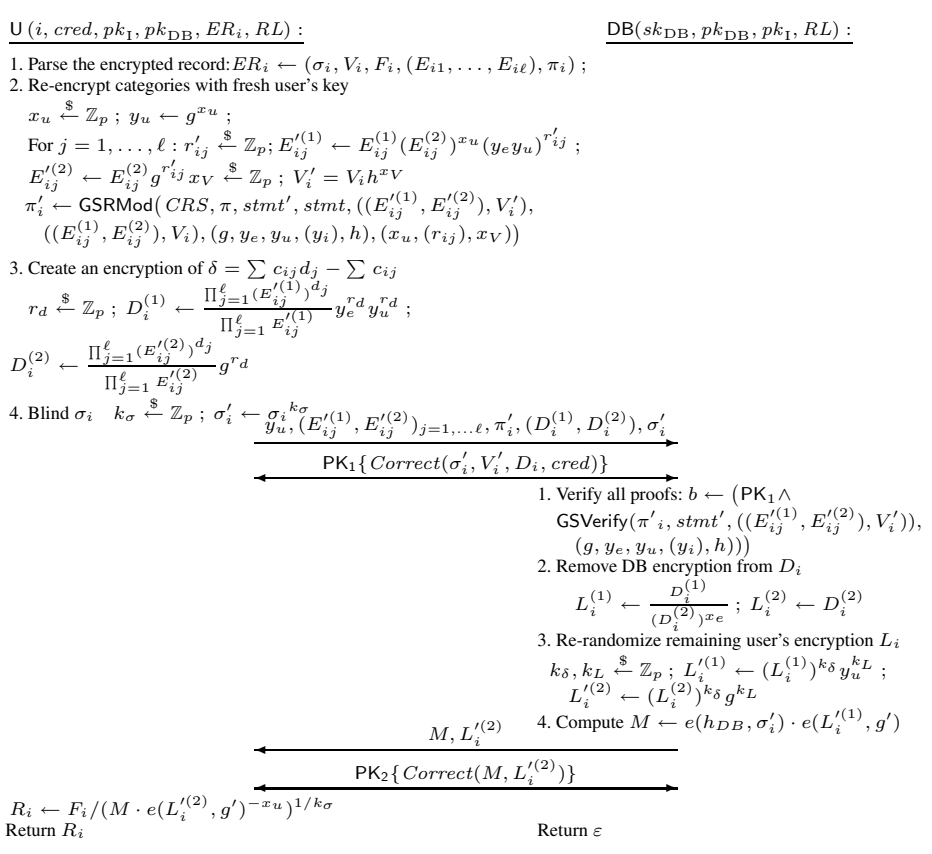

Fig. 5. The Transfer() protocol. The details of the proof protocols PK_1 and PK_2 are described in the text, as are the definitions of the statements $stmt$ and $stmt'$ of the GS proofs. The latter essentially say that the encryptions (E_{ij}) and (E'_{ij}) are consistent with V_i and V'_i, respectively.

Then, the user blinds σ_i and sends this blinded version σ'_i to the database server. Note that σ_i is derived from the database provider's secret key, the index of the records, and, most importantly, all the categories associated to the record. Next, the user proves to the database interactively that σ'_i is correctly formed as a randomization of some σ_i for which she possesses all necessary credentials, that V'_i is consistent with σ'_i, and that D_i is is correctly formed from the re-encrypted ACP and her credentials. That is, she executes with the database server the step we refer to as $\mathsf{PK}_1\{Correct(\sigma'_i, V'_i, D_i, cred)\}$ in Figure 5. For this proof, the user first searches in the current revocation list $RL = (t, R_1, R_2)$ for a tuple $(uid_{\text{left}}, uid_{\text{right}}, S, \hat{r}, \hat{s}) \in R_2$ such that $uid_{\text{left}} < uid < uid_{\text{right}}$. Let Δ_{left} and Δ_{right} such that $uid_{\text{left}} + \Delta_{\text{left}} = uid = uid_{\text{right}} - \Delta_{\text{right}}$ and let $y_\Delta^{(\Delta_{\text{left}})}$ and $y_\Delta^{(\Delta_{\text{right}})}$ be the issuer's signatures on these distances. The user next blinds $y_\Delta^{(\Delta_{\text{left}})}$ and $y_\Delta^{(\Delta_{\text{right}})}$ and her credentials, i.e., she computes $t_1, t'_1 \xleftarrow{\$} \mathbb{Z}_p$; $\tilde{A} \leftarrow Au^{t_1}$; $B \leftarrow w^{t_1} u^{t'_1}$, $t_2, t'_2 \xleftarrow{\$} \mathbb{Z}_p$; $\tilde{S} \leftarrow Su^{t_2}$, and $\tilde{S}_1 \leftarrow w^{t_2} u^{t'_2}$ to blind her credentials

and computes $t_3, t_3' \xleftarrow{\$} Z_p$; $\tilde{Y} \leftarrow (y_\Delta^{(\Delta_{\text{left}})})^{t_3}$; $\tilde{Y}_1 \leftarrow (y_\Delta^{(\Delta_{\text{right}})})^{t_3'}$ to blind the revocation distance signatures.

The user sends $\tilde{A}, B, \tilde{S}, \tilde{S}_1, \tilde{Y}$, and $\tilde{Y}_1$ to the database server and then executes the following proof of knowledge:

$$\mathsf{PK}_1\{(uid, uid_{\text{left}}, uid_{\text{right}}, k_\sigma, x_V, r_d, r, \alpha, \beta, \alpha_r, \beta_r, t_1, t_1', t_2, t_2', t_3, t_3', d_1, \ldots, d_\ell,$$

$$s, \hat{s}, \hat{r}, c_1, \ldots, c_\ell, v_i) : V_i' = g^i \prod_{i=1}^{\ell} y_i^{c_i} y_{\ell+1}^{v_i} h^{x_V} \wedge 1 = B^{-s} w^\alpha u^\beta \ \wedge B = w^{t_1} u^{t_1'}$$

$$\wedge\, e(y_{DB} V_i', \sigma_i') = e(g, g')^{k_\sigma} e(h, \sigma_i')^{x_V} \wedge \tilde{S}_1 = w^{t_2} u^{t_2'} \wedge 1 = \frac{w^{\alpha_r} u^{\beta_r}}{\tilde{S}_1^{\hat{s}}} \wedge$$

$$\frac{\overline{e}(\tilde{A}, y_I)}{\overline{e}(g_I, h_I)} = \overline{e}(\tilde{A}, h_I)^{-s} \overline{e}(u, y_I^{t_1} h_I^\alpha) \overline{e}(h_0, h_I)^{uid} \overline{e}(h_{\ell+1}, h_I)^r \prod_{k=1}^{\ell} \overline{e}(h_k, h_I)^{d_k} \wedge$$

$$\frac{\overline{e}(\tilde{S}, y_r)}{\overline{e}(\hat{g}, h_r)} = \overline{e}(\tilde{S}, h_r)^{-\hat{s}} \overline{e}(u, y_r^{t_2} h_r^{\alpha_r}) \overline{e}(\hat{g}_1, h_r)^t \overline{e}(\hat{g}_2, h_r)^{uid_{\text{left}}} \overline{e}(\hat{g}_3, h_r)^{uid_{\text{right}}} \overline{e}(\hat{g}_4, h_r)^{\hat{r}}$$

$$\wedge\, \overline{e}(\tilde{Y}, y_\Delta) = \overline{e}(\tilde{Y}, h_\Delta)^{-(uid - uid_{\text{left}})} \overline{e}(g_I, h_\Delta)^{t_3} \wedge$$

$$\overline{e}(\tilde{Y}_1, y_\Delta) = \overline{e}(\tilde{Y}_1, h_\Delta)^{-(uid_{\text{right}} - uid)} \overline{e}(g_I, h_\Delta)^{t_3'} \wedge$$

$$D_i^{(1)} \cdot \prod_{j=1}^{\ell} E_{ij}'^{(1)} = \prod_{j=1}^{\ell} (E_{ij}'^{(1)})^{d_j} y_e^{r_d} y_u^{r_d} \wedge D_i^{(2)} \cdot \prod_{j=1}^{\ell} E_{ij}'^{(2)} = \prod_{j=1}^{\ell} (E_{ij}'^{(2)})^{d_j} g^{r_d} \}$$

If the proof is successful, the database removes its layer of encryption from D_i and then randomizes the remaining encryption of δ (using the user's temporary public key y_u). This ensures that if $\delta \neq 0$ holds, then the encryption will contain a random value and is not related to the decryption key for the record. The database then proves to the user that it had computed everything correctly by executing with with her the protocol we referred to as $\mathsf{PK}_2\{Correct(M, L_i'^{(2)})\}$ in Figure 5. More precisely, it is the following proof that the values M and $L_i'^{(2)}$ were computed correctly (whereby $\gamma = -x_e k_\delta$):

$$\mathsf{PK}\{(h_{DB}, x_e, k_L, k_\delta, \gamma) : H = e(h_{DB}, g') \wedge y_e = g^{x_e} \wedge L_i'^{(2)} = (D_i^{(2)})^{k_\delta} g^{k_L} \wedge$$

$$1 = y_e^{k_\delta} g^\gamma \wedge M = e(h_{DB}, \sigma_i') \cdot e(D_i^{(1)}, g')^{k_\delta} \cdot e(D_i^{(2)}, g')^\gamma e(g, g')^{k_L} \} \ .$$

When the user gets L from the database, removes all randomness and decrypts, the decrypted value R_i is correct if and only if $\delta = 0$.

We finally remark that the database has to calculate encryptions of all ACPs and encrypt all records $(1, \ldots, N)$ only once at the setup phase, and the user has to download and verify the entire encrypted database only once as well. So the communication and computation complexity of the protocol depend on the number of the records in the database only in the setup and verify phases. The other parts of the protocol (issue and transfer) require only $O(\ell)$ group elements to be sent and exponentiations and pairings to be computed, where ℓ is the size of ACP vector.

5 Security Analysis

The security of our protocol is analyzed by proving indistinguishability between adversary actions in the real protocol and in an ideal scenario that is secure by definition.

Given a real-world adversary A, we construct an ideal-world adversary A' such that no environment $\mathcal{E}$ can distinguish whether it is interacting with A or A'. We organize the proof in sub-lemmas according to which subset of parties are corrupted. We do not consider the cases where all parties are honest, where all parties are dishonest, where the issuer is the only honest party, or where the issuer is the only dishonest party, as these cases have no real practical interest.

For each case we prove the indistinguishability between the real and ideal worlds by defining a sequence of hybrid games $\mathbf{Game\text{-}0}, \ldots, \mathbf{Game\text{-}n}$. In each game we define a simulator Sim_i that runs A as a subroutine and that provides $\mathcal{E}$'s entire view. We define $\mathbf{Hybrid}_{\mathcal{E},\mathsf{Sim}_i}(\kappa)$ to be the probability that $\mathcal{E}$ outputs 1 when run in the world provided by Sim_i. The games are always constructed such that the first simulator Sim_0 runs A and all honest parties exactly like in the real world, so that $\mathbf{Hybrid}_{\mathcal{E},\mathsf{Sim}_0}(\kappa) = \mathbf{Real}_{\mathcal{E},\mathsf{A}}(\kappa)$, and such that the final simulator Sim_n is easily transformed into an ideal-world adversary A' so that $\mathbf{Hybrid}_{\mathcal{E},\mathsf{Sim}_n}(\kappa) = \mathbf{Ideal}_{\mathcal{E},\mathsf{A}'}(\kappa)$. By upper-bounding and summing the mutual game distances $\mathbf{Hybrid}_{\mathcal{E},\mathsf{Sim}_i}(\kappa) - \mathbf{Hybrid}_{\mathcal{E},\mathsf{Sim}_{i+1}}(\kappa)$ for $i = 0, \ldots, n-1$, we obtain an upper bound for the overall distance $\mathbf{Real}_{\mathcal{E},\mathsf{A}}(\kappa) - \mathbf{Ideal}_{\mathcal{E},\mathsf{A}'}(\kappa)$.

Theorem 2. *If the $(N+2)$-BDHE and XDDH assumptions hold in $\mathbb{G}_1, \mathbb{G}_T$, the $(N+1)$-SDH assumption holds in $\mathbb{G}_1$, and M-SDH assumption holds in $\overline{\mathbb{G}}_1$ then the HAC-OT protocol in Section 4 securely implements the HAC-OT functionality, where N is the number of database records, and M is the number of the users.*

We prove the theorem by separately proving it for all relevant combinations of corrupted parties in four lemmas.

Due to lack of space, the detailed proof is found in the full version of this paper.

6 Conclusion

We have provided a set of efficient protocols and thereby shown that it is possible to build an access control system for a database with the maximal possible privacy for all involved parties: users can access records they are authorized to access without the server obtaining any information whatsoever about which records they access, which authorizations they users have, or whether the access was successful. Indeed, the database only learns that some user attempted to access the database. At the same time the database server is guaranteed that users can only access a single record and do not get any other information including information about other records or any access control list. Indeed, the user only learn whether or not their current credentials are sufficient to access the record they try to access.

The protocols we provide are fairly practical and we believe applications where records are relatively valuable, e.g., keys to decrypt some media such as movies or

a particular DNA-sequence of many people, then our protocol could be used in practice. Still, it remains an open question whether more efficient protocols exist. One way to achieve this, could be using attribute based encryption instead of using anonymous credentials. Our initial investigation of such protocols makes us believe that such an approach would lead to less efficient protocols. Nevertheless, further research along this lines could be fruitful.

Acknowledgements

The authors thank the anonymous referees of CCS 2009 for suggesting the problem of oblivious transfer with hidden access control lists. We are grateful to Robert Enderlein for saving us from a few embarrassing typos. This work was supported by the European Community through the Seventh Framework Programme (FP7/2007-2013) project PrimeLife (grant agreement no. 216483).

References

1. Au, M.H., Susilo, W., Mu, Y.: Constant-size dynamic *k*-TAA. In: De Prisco, R., Yung, M. (eds.) SCN 2006. LNCS, vol. 4116, pp. 111–125. Springer, Heidelberg (2006)
2. Boneh, D., Boyen, X., Shacham, H.: Short group signatures. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 41–55. Springer, Heidelberg (2004)
3. Belenkiy, M., Camenisch, J., Chase, M., Kohlweiss, M., Lysyanskaya, A., Shacham, H.: Randomizable proofs and delegatable anonymous credentials. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 108–125. Springer, Heidelberg (2009)
4. Bradshaw, R.W., Holt, J.E., Seamons, K.E.: Concealing complex policies with hidden credentials. In: 11th (CCS 2004), pp. 46–157. ACM Press, New York (2004)
5. Canetti, R.: Studies in Secure Multiparty Computation and Applications. PhD thesis, Weizmann Institute of Science, Rehovot 76100, Israel (June 1995)
6. Canetti, R.: Security and composition of multi-party cryptographic protocols. Journal of Cryptology 13(1), 143–202 (2000)
7. Camenisch, J., Chandran, N., Shoup, V.: A public key encryption scheme secure against key dependent chosen plaintext and adaptive chosen ciphertext attacks. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 351–368. Springer, Heidelberg (2009)
8. Camenisch, J., Dubovitskaya, M., Neven, G.: Oblivious transfer with access control. In: ACM CCS 2009, pp. 131–140. ACM Press, New York (2009)
9. Coull, S., Green, M., Hohenberger, S.: Controlling access to an oblivious database using stateful anonymous credentials. In: Jarecki, S., Tsudik, G. (eds.) PKC 2009. LNCS, vol. 5443, pp. 501–520. Springer, Heidelberg (2009)
10. Camenisch, J., Lysyanskaya, A.: Signature schemes and anonymous credentials from bilinear maps. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 56–72. Springer, Heidelberg (2004)
11. Camenisch, J., Neven, G., Shelat, A.: Simulatable adaptive oblivious transfer. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 573–590. Springer, Heidelberg (2007)
12. Dodis, Y., Haralambiev, K., Lopez-Alt, A., Wichs, D.: Cryptography against continuous memory attacks. Cryptology ePrint Archive, Report 2010/196 (2010)
13. Dodis, Y., Yampolskiy, A.: A verifiable random function with short proofs and keys. In: Vaudenay, S. (ed.) PKC 2005. LNCS, vol. 3386, pp. 416–431. Springer, Heidelberg (2005)

14. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory 31(4), 469–472 (1985)
15. Frikken, K.B., Atallah, M.J., Li, J.: Attribute-based access control with hidden policies and hidden credentials. IEEE Trans. Computers 55(10), 1259–1270 (2006)
16. Groth, J., Sahai, A.: Efficient non-interactive proof systems for bilinear groups. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 415–432. Springer, Heidelberg (2008)
17. Holt, J.E., Bradshaw, R.W., Seamons, K.E., Orman, H.K.: Hidden credentials. In: ACM WPES 2003, USA, pp. 1–8. ACM, New York (2003)
18. Herranz, J.: Restricted adaptive oblivious transfer. Cryptology ePrint Archive, Report 2008/182 (2008)
19. Li, N., Winsborough, W.: Towards practical automated trust negotiation. In: POLICY 2002, Washington, DC, USA, p. 92. IEEE Computer Society, Los Alamitos (2002)
20. Nakanishi, T., Fujii, H., Hira, Y., Funabiki, N.: Revocable group signature schemes with constant costs for signing and verifying. In: Jarecki, S., Tsudik, G. (eds.) PKC 2009. LNCS, vol. 5443, pp. 463–480. Springer, Heidelberg (2009)
21. Naor, M., Pinkas, B.: Oblivious transfer with adaptive queries. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 573–590. Springer, Heidelberg (1999)
22. Pfitzmann, B., Waidner, M.: Composition and integrity preservation of secure reactive systems. In: 7th ACM CCS, pp. 245–254. ACM Press, New York (2000)
23. Pfitzmann, B., Waidner, M.: A model for asynchronous reactive systems and its application to secure message transmission. In: Proceedings of the IEEE Symposium on Research in Security and Privacy, pp. 184–200. IEEE Computer Society, IEEE Computer Society Press, Los Alamitos (2001)
24. Rabin, M.O.: How to exchange secrets by oblivious transfer. Technical Report TR-81, Harvard Aiken Computation Laboratory (1981)
25. Yao, A.C.: Protocols for secure computations. In: 23rd FOCS, pp. 160–164. IEEE Computer Society Press, Los Alamitos (1982)
26. Yu, T., Winslett, M.: A unified scheme for resource protection in automated trust negotiation. In: IEEE Symposium on Security and Privacy (S&P 2003), USA, pp. 110–122. IEEE Computer Society, Los Alamitos (2003)
27. Yu, T., Winslett, M., Seamons, K.E.: Interoperable strategies in automated trust negotiation. In: ACM CCS 2001, pp. 146–155. ACM Press, New York (2001)
28. Aiello, W., Ishai, Y., Reingold, O.: Priced oblivious transfer: How to sell digital goods. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, p. 119. Springer, Heidelberg (2001)