Oblivious Transfer with Hidden Access Control from Attribute-Based Encryption [*]

Jan Camenisch[1], Maria Dubovitskaya[1,2], Robert R. Enderlein[1,2], and Gregory Neven[1]

[1] IBM Research – Zurich, Säumerstrasse 4, CH-8803 Rüschlikon, Switzerland
[2] Department of Computer Science, Swiss Federal Institute of Technology (ETH Zürich), CH-8092 Zürich, Switzerland

Abstract. The notion of oblivious transfer with hidden access control policies (HACOT) was recently proposed by Camenisch et al. (Public-Key Cryptography 2011). This primitive allows a user to anonymously query a database where each record is protected by a hidden attribute-based access control policy. At each query, the user either learns the value of a single record if the attributes in his key satisfy the policy, or the mere fact that his attributes do not satisfy the policy. The database, even when colluding with the key issuer, learns nothing about the identity of the user, the index or the access policy of the record, or whether access was granted or denied. At the same time, the database can keep an eye on the overall access frequency to prevent the data from being "crawled".

In this paper, we present a new HACOT scheme which is more efficient and offers more expressive policies than the scheme presented by Camenisch et al. We construct our HACOT protocol based on a hidden ciphertext-policy attribute-based encryption (HP-ABE) scheme by Nishide et al.: users are issued HACOT decryption keys based on HP-ABE attributes and HACOT records are encrypted under HP-ABE policies. However, as we will see, this simple approach does not work and we need to extend the Nishide et al. scheme as follows. First, we add protocols that allows users to verify that the public key of the issuer and ciphertexts are correctly formed. Second, we reserve one attribute and give the corresponding decryption key only to the database. Thereby users can no longer decrypt records by themselves but require the help of the database. Third, we provide a joint decryption protocol between the user and the database, so that the database does not learn which ciphertext is decrypted. The latter will also allow one to optionally add revocation of the users' access. We prove our construction secure by a reduction to the security of Nishide et al.'s scheme, the Symmetric External Diffie-Hellman (SXDH) and Simultaneous Flexible Pairing (SFP) assumptions.

Keywords: Privacy, Oblivious Transfer, Attribute-Based Encryption.

[*] This is the full version of a paper due to appear at the 8th Conference on Security and Cryptography for Networks (SCN 2012).

1 Introduction

Consider a medical database containing patients' medical records. Clearly, proper encryption and access control mechanisms need to be in place to protect such sensitive data. Different access control policies may apply to different records, ensuring for example that only relevant specialists and the treating medical staff have access to a patient's record. Given the frequent changes in medical personnel at hospitals, a role-based or attribute-based approach seems a natural solution.

Mere access control may not be enough, however. First, the access control policy by itself may leak sensitive information about a patient's illness. For example, the nature of a patient's health problem is pretty clear if an oncologist, a psychiatrist, or a plastic surgeon has access to his or her record. The treating medical staff may also have an interest in hiding the access control policies, e.g., to avoid being approached by the press when treating celebrities. Second, the query pattern for a particular record may reveal considerable information about the seriousness of the patient's condition or the phase of treatment. It is therefore desirable that the database can be queried anonymously, so that the database administrator remains oblivious as to who accesses which record at which time. Third, the database owner may want to prevent its users from abusing their anonymity to "crawl" the database and re-create a copy outside the owner's control. The database must therefore be able to detect unusual query activity to throttle requests, e.g., by inserting time delays or by presenting CAPTCHAs.

Narayan et al. [NGSN10] proposed a privacy-preserving Electronic Health Records (EHR) system that allows one to share patient data among healthcare providers in the cloud, using attribute-based encryption. However, the cloud provider still learns which files get downloaded (the scheme does not provide oblivious access to the data), the access control policy of records is not hidden, and revocation only works when all data is deleted from the user's device after each query.

A full-fledged solution is given by Camenisch et al. [CDNZ11], who combine adaptive oblivious transfer (OT), anonymous credentials, and zero-knowledge proofs to build a primitive called Oblivious Transfer with Hidden Access Control Policies (HACOT).

As observed by Camenisch et al. [CDNZ11], attribute-based encryption with hidden ciphertext policies (HP-ABE) [KSW08,NYO08,LOS$^+$10] is a similar primitive: a user's decryption key is associated with a list of attributes, while a ciphertext is associated with a hidden access control policy so that it can only be decrypted by users whose attributes satisfy the policy. Thus one could attempt to apply an HP-ABE scheme in the scenario above: issuing decryption keys to the medical personnel, encrypting each patient's record under the appropriate policy, and sending all ciphertexts to all users, all mentioned security requirements are met. However, this approach does not offer all the necessary security features, as we will discuss in detail in this paper. For example, it allows users to crawl the database and "bulk-decrypt" all records to which they have access offline. Also, it is not possible to revoke access, an indispensable feature in a changing environment such as a hospital's workforce. Revocation is notoriously difficult to implement in identity-based and attribute-based encryption systems, with most practical solutions requiring a painful trade-off to be made between security and frequency of key updates and database re-encryptions. Finally, as we will explain in more detail later, a HP-ABE scheme does not exclude that a single ciphertext decrypts to different plaintexts for different users, leading to anonymity problems. So, a plain application of HP-ABE is not satisfactory and, indeed, the construction of an HACOT scheme based on HP-ABE was left as an open problem by Camenisch et al. [CDNZ11].

Our Contributions. In this paper, we extend the HP-ABE scheme by Nishide et al. [NYO08,Nis08] as follows into an HACOT scheme to bring it to the same level of functionality as the HACOT protocol of [CDNZ11]. First, we add protocols that allow users to verify that the public key of the issuer and ciphertexts are correctly formed. Second, we reserve one attribute and give the corresponding decryption key only to the database so that users can no longer decrypt records by themselves but require the help of the database. Instead, we provide a joint decryption protocol between the user and the database, so that users can again decrypt records under the control of the database but with the database not learning which particular ciphertext is decrypted. The latter will also allow to optionally add revocation of the users' access. Thus, when using our functionality/scheme with an authenticated encryption scheme [BN08], one indeed obtains the required solution.

We also address a deficiency in the definition and protocol by Camenisch et al. [CDNZ11]: in their ideal functionality, users are returned $\perp$ if they do not have the necessary access rights, but in their protocol, they receive a random message in this case. We address this as follows: we define the ideal functionality 1) to handle only encryption keys that can then be used to derive keys for a symmetric encryption scheme and 2) in case a user has no access, the functionality will return a random key to the user.

Our construction relies on interactive zero-knowledge proofs of knowledge [CDM00], Groth-Sahai non-interactive proofs [GS08], the privacy-friendly signature scheme by Abe et al. [AFG$^+$10] and an HP-ABE scheme that allows transformations described in Section 4. Our HACOT protocol offers several advantages over that of Camenisch et al. [CDNZ11]. First, access control policies in our protocol are specified as vectors of subsets of polynomial-size attribute universes, which is more expressive than the boolean attributes of their scheme. Communication and computation by the database in the decryption protocol is independent of the number of attributes in our scheme, versus linear in theirs. Indeed, already for 3 attributes all operations are more efficient in our scheme except the key generation where ours is costly (using realistic security parameters for both schemes).

We prove our construction secure in the common reference string (CRS) model under the Symmetric eXternal Diffie-Hellman (SXDH) assumption, the Simultaneous Flexible Pairing (SFP) [AFG$^+$10] assumption, and the security of the underlying HP-ABE scheme (in the case of Nishide et al.'s scheme: generic bilinear group model). We also implemented a prototype of our protocol and provide a theoretical efficiency analysis and experimental performance results. Notice that database updates are also supported in our scheme, and records can be added to the database without having to re-encrypt the whole database. The database provider just distributes an update containing the ciphertexts of the new records.

Related Work. All attribute-based [SW05,BSW07,NYO08,LOS$^+$10] and predicate [KSW08,LOS$^+$10] encryption schemes allow for offline decryption, and can therefore not be used as such for the scenario we envisage. Some of these schemes [KSW08,NYO08,LOS$^+$10] are *policy-hiding*, meaning that users cannot deduce anything about the policy of a ciphertext except whether their key satisfies it. The schemes of Katz et al. and Lewko et al. [KSW08,LOS$^+$10] allow for conjunctions and disjunctions in the policy, but require composite-order bilinear groups. The scheme of Nishide et al. [NYO08] allows for slightly less powerful policies but works in a prime-order group setting. The scheme of Lewko et al. [LOS$^+$10] and the second construction of Nishide et al. [NYO08] are fully secure in the generic group model [Nis08], whereas the other schemes are only selectively secure.

There is an extensive body of literature on the subject of oblivious transfer [Rab81]. In this paper we use the adaptive k-out-of-n variant [NP99,CNS07], where a user may query for up to k records from a database of n records. The user does not learn anything about the $(n-k)$ records he did not query, while the database does not learn which records were queried. The goal is to "amortize" communication costs so that the encrypted database of size linear in n is transmitted once, but each transfer afterwards has communication cost independent of n. Several extensions to adaptive k-out-of-n OT have been proposed, including pricing [CDN10] and access control with known [CDN09,CGH09] and hidden [CDNZ11] policies. Our protocol is an alternative instantiation of the latter primitive.

Zhang et al. [ZAW$^+$10] have proposed a scheme that combines the OT protocol of Camenisch el al. [CNS07] with the attribute based encryption scheme of Lewko et al. [LOS$^+$10] and achieves oblivious transfer with access control functionality, but their scheme does not cover the hidden policy case and does not provide revocation of users. We also show in Section 4 that this scheme cannot be extended to a hidden policy while preserving our HACOT security properties.

Green et al. [GHW11] propose a way to outsource the main computation for decrypting ABE ciphertexts to a (possibly passively malicious) proxy, however their approach does not fit well with our real-world/ideal-world security notions since it is not clear what guarantees the user has if the proxy maliciously deviates from its specifications.

2 Definitions

An oblivious transfer protocol with hidden access control policies (HACOT) is run between an *issuer*, who sets up the system, and generates keys of users; one or more *databases*, who publish records and control users' access rights to the database by setting access policies; and *users*, who anonymously fetch records that they are entitled to access. Let $\mathcal{I}$ denote the issuer, $\mathcal{DB}$ the set of all databases, $\mathcal{DB}_\varrho$ the database with index ϱ, $\mathcal{U}$ the set of all users, $\mathcal{U}_\varphi$ the user with index φ.

2.1 Syntax and Basic Terminology

By $\mathbb{N}$ we denote the set of natural numbers, by $\mathbb{N}_n$ the set of all natural numbers between 0 and $(n-1)$. By $\mathbb{Z}_p$ we denote the ring of integers modulo p. We use $\mathbb{N}_n^*$ and $\mathbb{Z}_p^*$ to denote $\mathbb{N}_n \setminus \{0\}$ and $\mathbb{Z}_p \setminus \{0\}$, respectively. By $\mathrm{e}\colon \mathbb{G}_1 \times \mathbb{G}_2 \mapsto \mathbb{G}_\mathrm{T}$ we denote a bilinear map. If $\kappa \in \mathbb{N}$, then $\mathbf{1}^\kappa$ denotes the string consisting of κ ones.

If $\mathbb{A}$ is a set, then $a \overset{\$}{\leftarrow} \mathbb{A}$ means we set a to a random element of that set. If $\mathcal{A}$ is a Probabilistic Polynomial-Time (PPT) algorithm or interactive machine, then $y \overset{\$}{\leftarrow} \mathcal{A}(x)$ means we assign y to the output of $\mathcal{A}$ when run with fresh random coins on input x. If $\mathcal{A}$ and $\mathcal{B}$ are two interactive machines, then let $(\mathit{In}_\mathcal{A}||\mathit{In}_\mathcal{B}) \to (\mathit{Out}_\mathcal{A}||\mathit{Out}_\mathcal{B})$ denote the sets of inputs and outputs of $\mathcal{A}$ and $\mathcal{B}$ during the interaction between these two machines.

2.2 Record Policies and User Attributes

Our HACOT scheme uses the same policy framework as the HP-ABE scheme by Nishide et al. [NYO08]. We recall the notation below for convenience.

All records in the database are encrypted. The database $\mathcal{DB}_\varrho$ can specify an access policy for an encrypted record $C_{\varrho,\psi}$, which is called a ciphertext policy ($W_{\varrho,\psi}$). The issuer gives $\mathcal{U}_\varphi$ a secret decryption key corresponding to the set of access attributes (L_φ) granted to the user.

We denote languages for the attributes and policies as $\mathcal{L}_L$ and $\mathcal{L}_W$ respectively. We write $L \models W$ to mean that W is satisfied by L, and $L \not\models W$ if not (i.e., a user key corresponding to L, respectively can and cannot decrypt a ciphertext with policy W).

We now describe the structure of attributes and policies in more detail. The keys issued to users are associated with a list of attributes $L = (L_1, \ldots, L_n)$ from n different categories. Let $n_i \in \mathbb{N}$ be the (finite and polynomial) number of possible attribute values in the i-th category. Without loss of generality, we can encode the n_i attributes of category i as elements of $\mathbb{N}_{n_i}$, so that $L_i \in \mathbb{N}_{n_i}$. For example, in a hospital scenario, the $n = 3$ categories could be (Job Title, Department, Gender), where Job Title can take any of the $n_1 = 5$ attributes {student, nurse, doctor, surgeon, administration}, Department can take any of the $n_2 = 4$ attributes {cardiology, maternity, neurology, oncology}, and Gender can be any of the $n_3 = 2$ attributes {male, female}.

Each record in the database has an access control policy associated with it. A policy $W = (W_1, \ldots, W_n)$ is expressed as a list of n subsets of attributes $W_i \subseteq \mathbb{N}_{n_i}$. A key endowed with the attribute list L is authorized to access a record if and only if all attributes in the key are also in the ciphertext policy, i.e., $L \models W \Leftrightarrow \forall i \in \mathbb{N}_{n+1}^* : L_i \in W_i$. For example, Alice may be a surgeon in the oncology department, so that her key is associated with (surgeon, oncology, female), while Bob, who is an administrative assistant in the maternity department, has key (administration, maternity, male). If a patient's medical record is protected by a policy W =({doctor, surgeon}, {cardiology, oncology}, {male, female}), then Alice can access the record, but Bob cannot.

One could view the ciphertext policy as implementing a limited version of conjunctive normal form: within a category, the policy specifies an OR condition on the attribute the user key has for that category; and all attributes of the key have to be in the access structure, basically an AND condition. Note that the access structure is hidden from the user, meaning that he cannot recover it from the ciphertext alone.

2.3 Definition of HACOT without Revocation

A HACOT scheme is a tuple of the following eight PPT algorithms and protocols:

- $\mathsf{IssuerSetup}(\mathcal{L}_L, \mathcal{L}_W) \overset{\$}{\to} (pk_\mathcal{I}, sk_\mathcal{I})$. This algorithm generates the system-wide issuer public key $pk_\mathcal{I}$ and corresponding secret key $sk_\mathcal{I}$. The input to this algorithm is a description of the set of attributes $\mathcal{L}_L$ that keys can be endowed with, and a description of the set of ciphertext policies $\mathcal{L}_W$.

- $\mathsf{VerifyIssuerKey}\big((pk_{\mathcal{I}})||(pk_{\mathcal{I}}, sk_{\mathcal{I}})\big) \xrightarrow{\$} (b||\varepsilon)$. Upon receiving the public key of the issuer, each user and each database runs this protocol with the issuer, so that the latter can prove that the issuer keys are correctly formed. The common input is the issuer's public key, the issuer's private input is his secret key. The output is a bit b indicating whether the user or database accepts the issuer's key.
- $\mathsf{DBSetup}(pk_{\mathcal{I}}) \xrightarrow{\$} (pk_{\mathcal{DB}_\varrho}, sk_{\mathcal{DB}_\varrho})$. The database with index ϱ runs this algorithm to generate its public key $pk_{\mathcal{DB}_\varrho}$ and corresponding private key $sk_{\mathcal{DB}_\varrho}$.
- $\mathsf{VerifyDBKey}\big((pk_{\mathcal{I}}, pk_{\mathcal{DB}_\varrho})||(pk_{\mathcal{I}}, pk_{\mathcal{DB}_\varrho}, sk_{\mathcal{DB}_\varrho})\big) \xrightarrow{\$} (b||\varepsilon)$. Upon receiving the public key of $\mathcal{DB}_\varrho$, each user runs this protocol with the database $\mathcal{DB}_\varrho$, so that the latter can prove in zero-knowledge that it knows the secret key corresponding to its public key. The common input consists of the issuer's and the database's public keys. The database's private input is its secret key. The output is a bit b indicating whether the public key $pk_{\mathcal{DB}_\varrho}$ is accepted.
- $\mathsf{IssueRecord}(pk_{\mathcal{I}}, pk_{\mathcal{DB}_\varrho}, sk_{\mathcal{DB}_\varrho}, K_{\varrho,\psi}, W_{\varrho,\psi}) \xrightarrow{\$} C_{\varrho,\psi}$. The database $\mathcal{DB}_\varrho$ runs this algorithm to publish a new record with index ψ. The input are the database's key pair, the issuer's public key, the plaintext $K_{\varrho,\psi} \in \mathbb{G}_{\mathrm{T}}$, and the ciphertext policy $W_{\varrho,\psi} \in \mathcal{L}_W$. The output is the ciphertext $C_{\varrho,\psi}$.
- $\mathsf{CheckRecord}(pk_{\mathcal{I}}, pk_{\mathcal{DB}_\varrho}, C_{\varrho,\psi}) \xrightarrow{\$} b$. Upon receiving a ciphertext $C_{\varrho,\psi}$, each user performs a check to test whether it is correctly formed. The output bit b indicates the result of that check.
- $\mathsf{Escrow}(sk_{\mathcal{I}}, pk_{\mathcal{I}}, pk_{\mathcal{DB}_\varrho}, C_{\varrho,\psi}) \xrightarrow{\$} (K_{\varrho,\psi}, W_{\varrho,\psi})$. With this algorithm, $\mathcal{I}$ can efficiently recover the plaintext and the policy of the ciphertext $C_{\varrho,\psi}$ without interacting with $\mathcal{DB}_\varrho$. This algorithm models the fact that in most HP-ABE systems, the issuer can recover a great deal of information from all ciphertexts by using his private key.
- $\mathsf{IssueUserKey}\big((pk_{\mathcal{I}}, \varphi, L_\varphi)||(pk_{\mathcal{I}}, sk_{\mathcal{I}}, \varphi, L_\varphi)\big) \xrightarrow{\$} (sk_{\mathcal{U}_\varphi}||\varepsilon)$. The user with index φ and the issuer run this interactive protocol to generate a new secret key $sk_{\mathcal{U}_\varphi}$ for $\mathcal{U}_\varphi$. We assume that the protocol is run over an authenticated channel. Common inputs are φ, attributes $L_\varphi \in \mathcal{L}_L$, and the public key of the issuer. The issuer has his secret key as private input. Only the user receives output from this protocol, namely his secret key $sk_{\mathcal{U}_\varphi}$.
- $\mathsf{Query}\big((sk_{\mathcal{U}_\varphi}, pk_{\mathcal{I}}, pk_{\mathcal{DB}_\varrho}, C_{\varrho,\psi})||(pk_{\mathcal{I}}, pk_{\mathcal{DB}_\varrho}, sk_{\mathcal{DB}_\varrho})\big) \xrightarrow{\$} (K'||\varepsilon)$. The user $\mathcal{U}_\varphi$ queries database $\mathcal{DB}_\varrho$ to attempt to decrypt record ψ. The common input contains the public keys of the issuer and database. The user's private input is his secret key and the ciphertext $C_{\varrho,\psi}$. (A more general definition would include the entire encrypted database in the user's input, but the most efficient schemes should not need this.) The database's secret input is its private key. Only the user receives output from this protocol, namely the recovered plaintext K'. If decryption was successful, then $K' = K_{\varrho,\psi}$, otherwise K' is a random element of $\mathbb{G}_{\mathrm{T}}$. Queries take place over anonymous channels so that the database does not know with which user it interacts.

We assume that the CRS and system parameters are generated according to the appropriate distributions and are made available to all participants.

This definition is similar to the definition of HACOT in [CDNZ11]. The differences are as follows: 1) our definition allows for more expressive policies; 2) records can be added individually instead of setting them up all at once at system startup; 3) our system allows for multiple independent databases; and 4) we explicitly model the level of access that the issuer has in our system though the Escrow functionality. With respect to the last point, our scheme is at a disadvantage compared to [CDNZ11]: while the issuer in [CDNZ11] can always generate for himself a user key that decrypts all records, he must interact with the database to recover the plaintext of a record; to recover the policy, one interaction with the database per attribute is necessary.

2.4 Security Definitions

To define the security of our protocol we take an approach from [CNS07,CDNZ11] which is inspired by universal composability [Can00] and reactive systems [PW00,PW01]. Namely, we prove a HACOT protocol secure through an indistinguishability argument between the instantiation of HACOT in the real world $\mathcal{R}_{\mathrm{HACOT}}$, where players run the set of cryptographic protocols, and an ideal world functionality $\mathcal{F}_{\mathrm{HACOT}}$, which applies the functionality the cryptographic protocols are supposed to realize.

In the real world after receiving a message from the environment $\mathcal{E}$ all parties run the corresponding cryptographic algorithms or engage in the protocols described in Section 2.3.

We briefly describe the $\mathcal{F}_{\mathrm{HACOT}}$ functionality below and provide a formal ideal world definition in Appendix B. We note that the interfaces between the environment $\mathcal{E}$ and $\mathcal{R}_{\mathrm{HACOT}}$ and between $\mathcal{E}$ and $\mathcal{F}_{\mathrm{HACOT}}$ are identical.

Ideal World (sketch). In the ideal world, $\mathcal{F}_{\mathrm{HACOT}}$ performs all actions only after relaying all received messages to the simulator $\mathcal{S}$ and getting an approval from $\mathcal{S}$. After receiving a first message from the issuer, $\mathcal{F}_{\mathrm{HACOT}}$ fixes the set of possible attributes and policies. It further relays all messages between users, databases and issuer during the Issuer and Database Setup phases and creates a list of all databases and users. Note that after Setup is completed, it is not possible to add users or databases. Later, during Record Issuance, $\mathcal{F}_{\mathrm{HACOT}}$ stores records and their policies upon request from the database. This can be repeated any number of times, meaning that the records can be added at any time. When receiving a key request from a user, $\mathcal{F}_{\mathrm{HACOT}}$ stores an entry with the set of attributes for that user's key. Finally, during a Query, $\mathcal{F}_{\mathrm{HACOT}}$ removes the user and record identifier from the user's request before forwarding it to $\mathcal{S}$ and to the corresponding database. After the database replies with a bit b, $\mathcal{F}_{\mathrm{HACOT}}$ checks whether the set of user's attributes satisfies the policy of the requested record; if $b = 0$, then it sends $\perp$ to the user; if $b = 1$ and the policy is satisfied, it sends the stored record back to the user, otherwise, it sends back a random group element.

Discussion of Security Properties. Informally, the specification of the ideal functionality $\mathcal{F}_{\mathrm{HACOT}}$ is such that the following security properties are trivially satisfied. Hence, any real-world implementation of the scheme must satisfy the same properties.

Database security. Users need to contact the database for each record that they want to access, so the database can keep an eye on the overall access frequency and throttle too-frequent requests. Users cannot determine whether their key satisfies the access policy of the record before the interaction. They cannot deduce anything about the contents of a record if they did not query it with a valid key that satisfies the policy. After a successful interaction, users cannot deduce anything about the policy except whether their key satisfies it or not. Cheating and colluding users cannot query any records that one of them could not have queried individually. In particular, they cannot "combine" or "rearrange" attributes in their keys.

User security. The only information that the database sees during a query is the mere fact that a query takes place. In particular, it cannot determine which user queries which record, which policy is associated to the record, which attributes the user has, and whether access to the record was granted or not. User security is valid even if the database colludes with the issuer and other users. If the query protocol completes successfully, honest users are guaranteed that 1) if access was granted, then their key satisfies the policy of the record and 2) if access was denied, then their key does not satisfy the policy of the record.

3 Preliminaries

In this section we describe the security assumptions and building blocks used in our scheme.

3.1 Assumptions

Decisional Diffie-Hellman (DDH) Assumption. Let $\mathbb{G}$ be either $\mathbb{G}_1$, $\mathbb{G}_2$ or $\mathbb{G}_{\mathrm{T}}$ (and let g be the corresponding generator of the group, viz. g_1, g_2, g_{T}). Let $a, b, z \overset{\$}{\leftarrow} \mathbb{Z}_p$. DDH is hard in $\mathbb{G}$ if for every PPT algorithm $\mathcal{A}$:

$$\mathrm{Adv}_{\mathbb{G}}^{\mathrm{DDH}} \overset{\mathrm{def}}{=} \left| \Pr\left[\mathcal{A}(g, g^a, g^b, g^{ab}) \overset{\$?}{=} 1\right] - \Pr\left[\mathcal{A}(g, g^a, g^b, g^z) \overset{\$?}{=} 1\right] \right| = \textit{negl}. \quad (1)$$

Symmetric External Diffie-Hellman (SXDH) Assumption. We say that the SXDH assumption holds if DDH holds in $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_{\mathrm{T}}$. This assumption holds only for type-3 [GPS08] bilinear maps: this means there exists no efficiently computable homomorphism from $\mathbb{G}_1$ to $\mathbb{G}_2$ or vice-versa. It is believed SXDH holds in certain subgroups of MNT elliptic curves.

ℓ-Simultaneous Flexible Pairing (SFP) Assumption. Let $A, B \xleftarrow{\$} \mathbb{G}_1$, $\tilde{A}, \tilde{B} \xleftarrow{\$} \mathbb{G}_2$ and $g_Z, f_Z, g_R, f_U \xleftarrow{\$} \mathbb{G}_1^*$. For $j \in \mathbb{N}_{\ell+1}^*$ let $P_j \overset{\text{def}}{=} \big(Z_j, R_j, S_j, T_j, U_j, V_j, W_j\big)$ that satisfies:

$$\begin{aligned} \mathrm{e}(A, \tilde{A}) &= \mathrm{e}(g_Z, Z_j)\, \mathrm{e}(g_R, R_j)\, \mathrm{e}(S_j, T_j) \wedge \\ \mathrm{e}(B, \tilde{B}) &= \mathrm{e}(f_Z, Z_j)\, \mathrm{e}(f_U, U_j)\, \mathrm{e}(V_j, W_j)\ . \end{aligned} \tag{2}$$

We say that the ℓ-SFP assumptions holds in $\mathbb{G}_1$ if for all PPT algorithms $\mathcal{A}$ (3) is satisfied [AFG$^+$10,AHO10], where $P_{\ell+1}$ satisfies (2) and $Z_{\ell+1} \neq 1 \wedge \forall i \in \mathbb{N}_{\ell+1}^* : Z_{\ell+1} \neq Z_i$.

$$\mathrm{Adv}_{\mathbb{G}_1}^{\ell-\mathrm{SFP}} \overset{\text{def}}{=} \Pr\left[\mathcal{A}\left(g_Z, f_Z, g_R, f_U, A, B, \tilde{A}, \tilde{B}, \left(P_j\right)_{j=1}^{\ell}\right) \overset{\$?}{=} P_{\ell+1}\right] = \textit{negl.} \tag{3}$$

We define ℓ-SFP in $\mathbb{G}_2$ analogously by exchanging the groups $\mathbb{G}_1$ and $\mathbb{G}_2$ above.

This assumption is parameterized by ℓ (unlike the other assumptions which are static), where a larger ℓ means a stronger assumption. This assumption was proven to hold in the generic bilinear group model [AHO10], as long as $\ell \ll \sqrt{p}$ (quadratic bound).

3.2 Zero-Knowledge Proofs

ZKPK denotes an interactive zero-knowledge proof (or argument) of knowledge [CDM00], while NIZK denotes a non-interactive zero-knowledge proof [GS08]. We will use the Camenisch-Stadler notation [CS97] to describe what is being proven, for example: $\mathsf{ZKPK}\{(\alpha, \beta) : y = g^\alpha \wedge z = g^\beta h^\alpha\}$. Variables in parenthesis denote the elements knowledge is proven about, such that the formula after the colon is true. Further details can be found in Appendix A.1 and A.2.

3.3 Hidden-Policy Attribute-Based Encryption

A HP-ABE scheme is a tuple of the following four PPT algorithms:

- $\mathsf{IssuerSetup}(1^\kappa, \mathcal{L}_L, \mathcal{L}_W) \xrightarrow{\$} (pkh_\mathcal{I}, skh_\mathcal{I})$. Using $\mathcal{L}_L, \mathcal{L}_W$, and a security parameter as input, this algorithm outputs the public and secret keys of $\mathcal{I}$.
- $\mathsf{IssueUserKey}(skh_\mathcal{I}, pkh_\mathcal{I}, L) \xrightarrow{\$} skh_\mathcal{U}$. Given a permissible set of attributes $L \in \mathcal{L}_L$ and the issuer's key pair, this generates a new user key $skh_\mathcal{U}$ endowed with L.
- $\mathsf{Encrypt}(pkh_\mathcal{I}, K, W) \xrightarrow{\$} C$. Given a plaintext $K \in \mathbb{G}_\mathrm{T}$, a permissible ciphertext policy $W \in \mathcal{L}_W$, and the issuer's public key $pkh_\mathcal{I}$, this algorithm generates a corresponding ciphertext C.
- $\mathsf{Decrypt}(skh_\mathcal{U}, pkh_\mathcal{I}, C) \to K'$. Given a ciphertext C, the user's secret key $skh_\mathcal{U}$, and the issuer's public key $pkh_\mathcal{I}$, this algorithm decrypts the ciphertext with the user's secret key. If the key satisfies the policy ($L \models W$), then the correct plaintext is recovered ($K' = K$). If the key does not satisfy the policy then $K' \neq K$ with overwhelming probability.

Security of HP-ABE. Informally, an HP-ABE scheme is secure if an adversary, who can adaptively get as many keys issued as he wants, cannot tell if a given challenge ciphertext decrypts to some plaintext M_0 under policy W_0 or to some other plaintext M_1 under W_1 of his choosing (modulo the trivial cases). A precise definition is given in Appendix A.5. Note that some HP-ABE schemes are only *selectively* secure (meaning the adversary must fix the challenge plaintexts and policies before he receives the issuer's key), but this is not sufficient for our scheme.

We use the second construction of Nishide et al.'s HP-ABE [NYO08] scheme. It is proven secure in the generic bilinear group setting [Nis08], and requires Type-3 pairings. It allows the issuer to add attributes and categories after system setup. In their scheme the issuer is assumed to be trusted, unlike our HACOT scheme.

3.4 Structure-Preserving Signatures

To hide the record index during a query, but at the same time make sure that the user is asking to decrypt a correct ciphertext, we need a signature scheme that allows for zero-knowledge proof-of-possession. As the ciphertext is a set of group elements, we use the basic signature scheme by Abe et al. [AFG$^+$10] for signing group elements. This scheme does not require the signer to know the discrete logarithm of the group elements he is signing. It is existentially unforgeable against adaptive chosen message attacks if the Simultaneous Flexible Pairing assumption [AFG$^+$10] holds.

This signature scheme Sig allows the user to re-randomize (blind) the signature and prove in zero-knowledge that this is still a correct signature. The user's anonymity is thus preserved, and the database has the guarantee that it is not helping the user to decrypt invalid ciphertexts. The scheme consists of a key generation algorithm Sig.KeyGen; a signing algorithm Sig.Sign; a verification algorithm Sig.Verify; a re-randomization algorithm Sig.Rerand, that takes as input a signature σ of a message m and outputs a re-randomized signature σ', which is also a valid signature on m; and two algorithms Sig.KeyProve and Sig.Prove for proving in zero-knowledge the validity of the key and the signature respectively. See Appendix A.3 for more details.

4 Achieving HACOT from HP-ABE

Let us give some intuition and a high-level description of our scheme, in particular how to extend a HP-ABE scheme into an oblivious access control system with hidden policies. In our protocol we use a concrete HP-ABE scheme by Nishide et al. [NYO08], but one can apply a similar trick to other HP-ABE schemes. Recall that in HP-ABE, a user's decryption key is associated with a list of attributes, while a ciphertext is associated with a hidden access control policy. A user can decrypt a ciphertext offline only if the attributes from his key satisfy the cipertext policy. Assume a database would just employ an HP-ABE scheme to encrypt all records and then publish these encryptions. Users would be issued the HP-ABE decryption keys corresponding to their attributes. In this approach, the access control policies would indeed be hidden and also users would only be able to access the records for which their attributes match the access control policies. Unfortunately, this solution does not provide all the properties that HACOT requires:

- First, somewhat counterintuitively, anonymity of the users is not guaranteed: in case the database and the issuer are malicious, they could deviate from the key issuance and encryption procedures so that if two users with the same attributes decrypt a record their result will still be different; hence, the scheme would not satisfy our ideal world definition. In a higher-level protocol, the two users might be distinguishable.
- Second, users can immediately decrypt all the records for which they have the necessary attributes. This will allow colluding users to derive information about the policies. Furthermore, the database has no control over the access frequency of records and cannot revoke access rights.

To address the first issue, we make the operations by the issuer and the database verifiable. We add an additional protocol (VerifyIssuerKey) in which the issuer proves that its keys were generated correctly, we turn the IssueUserKey algorithm that generates users' decryption keys into a two-party computation between the issuer and the user, and we provide a protocol VerifyEncryption allowing users to check that the ciphertexts are correctly formed.

Addressing the second issue is a bit trickier. A first idea to ensure that users cannot decrypt without the help of the database could be to combine a standard OT protocol to encrypt records twice, first under the OT protocol and then under the appropriate policies using the HP-ABE scheme. (Zhang et al. [ZAW$^+$10] used a similar approach to obtain an OT protocol with public access control policies.) This does not suffice, however, as users can perform the outer HP-ABE decryption step without interacting with the database and learn information about the access policies by observing whether decryption succeeds. Encrypting first under the HP-ABE scheme does not work either, since colluding users could start by doing a single OT query, and then attempt decrypting the inner HP-ABE ciphertext without oversight. Besides violating our ideal world definition, such a scheme excludes any interactive revocation scheme.

On a high level, our approach is the following. We create a dedicated "zeroth" attribute category of the HP-ABE scheme so that the issuer only issues decryption keys for one particular attribute $\mathcal{W}_{\mathcal{I}}$ in this zeroth category, but each database encrypts records under policies that require a different attribute $\mathcal{W}_{\mathcal{DB}_\varrho}$ for the zeroth category. The database has a "transformation key" that allows it to convert a ciphertext encrypted for $\mathcal{W}_{\mathcal{DB}_\varrho}$ into one for $\mathcal{W}_{\mathcal{I}}$. When the user wants to decrypt a record, she blinds the ciphertext and engages in a joint decryption protocol with

```
1. Generate HP-ABE key:
   $w, \beta \xleftarrow{\$} \mathbb{Z}_p^*$ ; $Y \leftarrow g_{\mathrm{T}}^{w}$ ; $B \leftarrow g_1^{\beta}$ ;
   $\{\{a_{i,t} \xleftarrow{\$} \mathbb{Z}_p^*$ ; $A_{i,t} \leftarrow g_1^{a_{i,t}}\}_{t=0}^{n_i-1}\}_{i=0}^{n}$ ;
   $skh_{\mathcal{I}} \leftarrow \big(\{a_{i,t}\}, w, \beta\big)$ ;
   $pkh_{\mathcal{I}} \leftarrow \big(\{A_{i,t}\}, Y, B\big)$.
2. Generate signing keys:
   $(sgk_{\mathcal{I}}, vk_{\mathcal{I}}) \xleftarrow{\$} \mathsf{Sig.KeyGen}(1^{\kappa})$ ;
Output $sk_{\mathcal{I}} \leftarrow (skh_{\mathcal{I}}, sgk_{\mathcal{I}})$ ;
       $pk_{\mathcal{I}} \leftarrow (pkh_{\mathcal{I}}, vk_{\mathcal{I}})$.
```

Fig. 1. IssuerSetup algorithm.

```
1. Generate HP-ABE keys for the
   DB-specific attribute:
   $k_{\varrho} \xleftarrow{\$} \mathbb{Z}_p$ ;
   $A_{0,\varrho} \leftarrow A_{0,0}^{k_{\varrho}} = g_1^{a_{0,0}k_{\varrho}}$ ;
   $skh_{\mathcal{DB}_{\varrho}} \leftarrow k_{\varrho}$ ; $pkh_{\mathcal{DB}_{\varrho}} \leftarrow A_{0,\varrho}$.
2. Generate signing keys:
   $(sgk_{\mathcal{DB}_{\varrho}}, vk_{\mathcal{DB}_{\varrho}}) \xleftarrow{\$} \mathsf{Sig.KeyGen}(1^{\kappa})$ ;
Output $sk_{\mathcal{DB}_{\varrho}} \leftarrow (skh_{\mathcal{DB}_{\varrho}}, sgk_{\mathcal{DB}_{\varrho}})$ ;
       $pk_{\mathcal{DB}_{\varrho}} \leftarrow (pkh_{\mathcal{DB}_{\varrho}}, vk_{\mathcal{DB}_{\varrho}})$.
```

Fig. 2. DBSetup algorithm.

the database to obtain a transformed ciphertext for $\mathcal{W}_{\mathcal{I}}$. To make sure that the user blinded a ciphertext that was previously published by the database, the database signs all of its ciphertexts. During the decryption protocol, the user proves knowledge of a valid signature for her blinded ciphertext—without revealing the signature, of course.

Now that the database is involved in the decryption process of the user, it becomes much easier to add revocation. Users are issued an anonymous credential and the database will only run the joint decryption protocol if the user's credential has not been revoked. There are a number of possible schemes to employ here and we will follow the choice by Camenisch et al. for their HACOT scheme [CDNZ11].

5 Our Construction Without Revocation

In this section we describe in detail how to construct our HACOT protocol (without the optional revocation mechanism) when instantiated with the second HP-ABE scheme by Nishide et al. [NYO08]. The description of the construction will be followed by a discussion about asymptotic complexity. The changes needed to handle revocation are described in Appendix C.

5.1 Detailed Construction

We now present the realizations of all algorithms and protocols of our scheme listed in Section 2.3 in detail.

System Parameters. We assume that the following parameters are generated by a trusted third party (or alternatively generated jointly via a multiparty computation) and are an extra (implicit) input to all algorithms and protocols. Concretely, all primitives we use require a common bilinear map setting: $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_{\mathrm{T}}, g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2, \mathrm{e}) \leftarrow \mathrm{Gen}(1^{\kappa})$. We denote $g_{\mathrm{T}} \stackrel{\mathrm{def}}{=} \mathrm{e}(g_1, g_2)$. For the Groth-Sahai proofs we also need a common reference string $\mathit{CRS} \leftarrow \{\mathfrak{U}_{1,2} \leftarrow g_2^{\mathfrak{a}}, \mathfrak{U}_{2,1} \leftarrow g_2^{\mathfrak{t}}, \mathfrak{U}_{2,2} \leftarrow g_2^{\mathfrak{a}\mathfrak{t}}\}$, where $\mathfrak{a}, \mathfrak{t} \xleftarrow{\$} \mathbb{Z}_p^*$.

Issuer Key Generation and Verification. In addition to the n regular categories, the issuer creates an additional zeroth category, and creates one attribute $\mathcal{W}_{\mathcal{I}}$ in that category. Let $A_{0,0} = g_1^{a_{0,0}}$ be the public key component associated to that attribute ($a_{0,0}$ is the private key component). All the users' keys he issues will contain this $\mathcal{W}_{\mathcal{I}}$ attribute (that is $L_0 = 0$). The key generation algorithm IssuerSetup is depicted in Figure 1, and takes as input the number of categories n and the number of attributes possible per category $\{n_i\}_{i=1}^{n}$; $n_0 = 1$.

Each party who receives the issuer's public key checks that the latter's key was generated correctly by running the VerifyIssuerKey protocol, which consists of checking that $Y \neq 1$, $B \neq 1$, $A_{i,t} \neq 1$, and running the following proof of knowledge with the issuer: $\mathsf{ZKPK}_1 \stackrel{\mathrm{def}}{=} \mathsf{ZKPK}\big\{\big(sgk_{\mathcal{I}}, w, \beta, \{\{a_{i,t}\}\}\big) : Y = g_{\mathrm{T}}^{w} \wedge B = g_1^{\beta} \wedge \bigwedge_{i=0}^{n}(\bigwedge_{t=0}^{n_i-1}(A_{i,t} = g_1^{a_{i,t}})) \wedge \mathsf{Sig.KeyProve}(sgk_{\mathcal{I}}, vk_{\mathcal{I}})\big\}$.

1. Encrypt the record with respect to the policy with the HP-ABE scheme:
   $\big(\hat{C}, C_0, C_{0,1}, C_{0,\varrho,2}, \{\{C_{i,t}\}_{t=0}^{n_i-1}\}_{i=1}^{n}\big) \xleftarrow{\$} \mathsf{Encrypt}(pkh_{\mathcal{I}}, K_{\varrho,\psi}, W_{\varrho,\psi})$, that is:
   Choose $\{r_i \xleftarrow{\$} \mathbb{Z}_p^*\}_{i=0}^{n}$; $\{\epsilon_{i,t} \xleftarrow{\$} \mathbb{Z}_p^*$ for all $t \in (\mathbb{N}_{n_i} \setminus W_i)\}_{i=1}^{n}$.
   Set $r = \sum_{i=0}^{n} r_i \bmod p$; $\{\epsilon_{i,t} \leftarrow 0$ for all $t \in W_i\}_{i=1}^{n}$; $\hat{C} \leftarrow KY^r$;
   $C_0 \leftarrow B^r$; $C_{0,\varrho,2} \leftarrow A_{0,\varrho}^{r_0}$; $\{C_{i,1} \leftarrow g_1^{r_i}\}_{i=0}^{n}$; $\{\{C_{i,t,2} \leftarrow A_{i,t}^{r_i} g_1^{\epsilon_{i,t}}\}_{t=0}^{n_i-1}\}_{i=1}^{n}$.
2. Generate a Groth-Sahai proof $\pi = \mathsf{NIZK}_3$ to assert correctness of the encryption:
   $\pi \overset{\text{def}}{=} \mathsf{NIZK}\{(\{r_i\}_{i=0}^{n}) : \bigwedge_{i=0}^{n} (C_{i,1} = g_1^{r_i}) \wedge C_0 = \prod_{i=0}^{n} B^{r_i} \wedge C_{0,\varrho,2} = A_{0,\varrho}^{r_0}\}$.
3. Sign the ciphertext component $C_{0,\varrho,2}$: $\sigma' \xleftarrow{\$} \mathsf{Sig.Sign}_{sgk_{\mathcal{DB}_\varrho}}(C_{0,\varrho,2})$.

Output $C_{\varrho,\psi} \leftarrow \big(\pi, \sigma', \hat{C}, C_0, C_{0,1}, C_{0,\varrho,2}, \{\{C_{i,t}\}_{t=0}^{n_i-1}\}_{i=1}^{n}\big)$.

Fig. 3. IssueRecord algorithm.

Database Key Generation and Verification. Recall that ϱ is the implicit database identifier. Each database $\mathcal{DB}_\varrho$ that joins the system extends the zeroth category with a new attribute, $\mathcal{W}_{\mathcal{DB}_\varrho}$. The public key component associated to that attribute is $A_{0,\varrho} = A_{0,0}^{k_\varrho}$, where k_ϱ is part of the database's private key. The value $A_{0,\varrho} = g_1^{a_{0,0}k_\varrho}$ can be considered as part of the public key of the "related" HP-ABE scheme, i.e., the secret key component corresponding to $\mathcal{W}_{\mathcal{DB}_\varrho}$ thereby implicitly becomes $a_{0,\varrho} = a_{0,0}k_\varrho \pmod{p}$.

The database setup consists of an algorithm DBSetup shown in Figure 2 to generate the database's key pair.

Each user receiving $pk_{\mathcal{DB}_\varrho}$ checks that the database's public was generated correctly with the protocol VerifyDBKey, which consists of first checking that $A_{0,\varrho} \neq 1$, and then running the following proof of knowledge with the database: $\mathsf{ZKPK}_2 \overset{\text{def}}{=} \mathsf{ZKPK}\big\{\big(sgk_{\mathcal{DB}_\varrho}, k_\varrho\big) : A_{0,\varrho} = A_{0,0}^{k_\varrho} \wedge \mathsf{Sig.KeyProve}(sgk_{\mathcal{DB}_\varrho}, vk_{\mathcal{DB}_\varrho})\big\}$.

Verifiable Encryption of Records. To encrypt a record with (implicit) index ψ, containing the plaintext $K \in \mathbb{G}_{\mathrm{T}}$ and the hidden ciphertext policy $W_{\varrho,\psi} = [W_0, \ldots, W_n]$, where $\forall i : W_i \subseteq \mathbb{N}_{n_i}$, the database runs IssueRecord as shown in Figure 3. If one wants to use messages $M \in \{0,1\}^*$ instead, one can use an authenticated encryption [BN08] algorithm AuthEnc that uses elements from $\mathbb{G}_{\mathrm{T}}$ as symmetric keys to encrypt M using the key K.

For the zeroth category, only the ciphertext component $C_{0,\varrho,2}$ for the $\mathcal{W}_{\mathcal{DB}_\varrho}$ attribute is published and not $C_{0,0,2}$ for the $\mathcal{W}_{\mathcal{I}}$ attribute. For the latter the users will have the decryption key but not for the former. However, as $C_{0,0,2} = (C_{0,\varrho,2})^{k_\varrho^{-1}}$, users can decrypt a record if (and only if) the database helps them.

Users need to verify all ciphertexts by running CheckRecord, which for each record verifies correctness of the GS proof π and the signature σ' on $C_{0,\varrho,2}$.

Issuing Decryption Keys to Users. Let $(L_1, \ldots, L_n)$, $L_i \in \mathbb{N}_{n_i}$, be the attributes of the user. We add to these $L_0 = 0$ (corresponding to $\mathcal{W}_{\mathcal{I}}$) and set $L_\varphi = (L_0, L_1, \ldots L_n)$. The protocol depicted in Figure 4 ensures that the keys are generated in an honest way, i.e., that the $D_{i,j}$'s are computed correctly with respect to L_i and contain a random λ_i. To this end, the user chooses an ephemeral ElGamal key pair and a random λ_i'' and sends the issuer encryptions of $g_2^{\lambda_i''}$. As discussed in Section 4, this will ensure the anonymity for the user during decryption. The issuer then chooses his own values of λ_i' and then computes the encryptions of the HP-ABE keys by modifying the received encryptions so that $\lambda_i \equiv \lambda_i' + \lambda_i'' \pmod{p}$ will hold. For this to work, the user and the issuer have to prove to each other that they did their computation correctly with the following two proof protocols. With the first protocol $\mathsf{ZKPK}_4 \overset{\text{def}}{=} \mathsf{ZKPK}\{(x, \{\lambda_i'', r_i\}_{i=1}^{n}) : X = g_2^x \wedge (\bigwedge_{i=1}^{n}(E_i = g_2^{\lambda_i''} X^{r_i} \wedge F_i = g_2^{r_i}))\}$ the user proves to the issuer that (E_i, F_i) is a valid encryption of the value $g_2^{\lambda_i''}$. With the second proof of knowledge $\mathsf{ZKPK}_5 \overset{\text{def}}{=} \mathsf{ZKPK}\{(w, \beta, s, \{\lambda_i', \tilde{a}_i\}_{i=0}^{n}, \{\tilde{r}_i\}_{i=1}^{n}) : Y = g_{\mathrm{T}}^w \wedge B = g_1^\beta \wedge 1 = g_2^w g_2^s (D_0^{-1})^\beta \wedge D_{0,2} = g_2^{\lambda_0''} g_2^{\lambda_0'} \wedge D_{0,1} = g_2^s D_{0,2}^{\tilde{a}_0} \wedge (\bigwedge_{i=0}^{n}(A_{i,L_i} = g_1^{\tilde{a}_i})) \wedge (\bigwedge_{i=1}^{n}(\tilde{E}_i = g_2^{\lambda_i'} E_i \wedge \hat{E}_i = g_2^s \tilde{E}_i^{\tilde{a}_i} X^{\tilde{r}_i} \wedge \hat{F}_i = F_i^{\tilde{a}_i} g_2^{r_{5,i}}))\}$ the issuer proves to the user that the encryptions he sent 1) were computed correctly and were based on the values he received from the user; 2) indeed encode the correct attributes, i.e., those defined by the A_{i,L_i}'s contained in the issuer's public

User φ $(pk_{\mathcal{I}}, \varphi, L_\varphi)$:	Authenticated channel	Issuer $(pk_{\mathcal{I}}, sk_{\mathcal{I}}, \varphi, L_\varphi)$:
$(\lambda_0'', x) \xleftarrow{\$} (\mathbb{Z}_p^*)^2$; $X \leftarrow g_2^x$;		$s \xleftarrow{\$} \mathbb{Z}_p^*$;
$\{\lambda_i'', r_i \xleftarrow{\$} (\mathbb{Z}_p)^2\}_{i=1}^n$;		$\{\tilde{a}_i \overset{\text{def}}{=} a_{i,L_i}$; $\tilde{r}_i \xleftarrow{\$} \mathbb{Z}_p\}_{i=1}^n$;
$\{(E_i, F_i) \leftarrow (g_2^{\lambda_i''} X^{r_i}, g_2^{r_i})\}_{i=1}^n$.	$\xrightarrow{X, \lambda_0'', \{E_i, F_i\}_{i=1}^n, \mathsf{ZKPK}_4}$	$\{\lambda_i' \xleftarrow{\$} \mathbb{Z}_p\}_{i=0}^n$.
		$\{\tilde{E}_i \leftarrow g_2^{\lambda_i'} E_i\}_{i=1}^n$; $D_{0,2} \leftarrow g_2^{\lambda_0' + \lambda_0''}$;
		$\{\hat{E}_i \leftarrow g_2^s \tilde{E}_i^{\tilde{a}_i} X^{\tilde{r}_i}$; $\hat{F}_i \leftarrow F_i^{\tilde{a}_i} g_2^{\tilde{r}_i}\}_{i=1}^n$;
		$\sigma'' \leftarrow \mathsf{Sig.Sign}_{sgk\mathcal{I}}(D_{0,2})$;
	$D_0, D_{0,1}, D_{0,2}, \sigma''$,	$D_0 \leftarrow g_2^{\frac{w+s}{\beta}}$; $D_{0,1} \leftarrow g_2^s D_{0,2}^{a_{0,0}}$.
$\{D_{i,1} \leftarrow \hat{E}_i \hat{F}_i^{-x}\}_{i=1}^n$;	$\xleftarrow{\{\tilde{E}_i, \hat{E}_i, \hat{F}_i\}_{i=1}^n, \mathsf{ZKPK}_5}$	
$\{D_{i,2} \leftarrow \tilde{E}_i F_i^{-x}\}_{i=1}^n$.		
Return $sk_{\mathcal{U}\varphi} \leftarrow (L_\varphi, \sigma'', D_0, \{D_{i,1}, D_{i,2}\}_{i=0}^n)$.		**Return** ε.

Fig. 4. Interactive issuing of a user's decryption keys. The proofs ZKPK_4 and ZKPK_5 are defined in the text.

User φ $(sk_{\mathcal{U}\varphi}, pk_{\mathcal{I}}, pk_{\mathcal{DB}_\varrho}, C_{\varrho,\psi})$:	Anonymous channel	Database ϱ $(pk_{\mathcal{I}}, pk_{\mathcal{DB}_\varrho}, sk_{\mathcal{DB}_\varrho})$:
Blind $C_{0,\varrho,2}$ of record ψ and $D_{0,2}$ of user φ :		
$k_c, k_d \xleftarrow{\$} \mathbb{Z}_p^*$; $C' \leftarrow C_{0,\varrho,2}^{k_c}$; $D'' \leftarrow D_{0,2}^{k_d}$.		
Re-randomize sigantures:		
$\tilde{\sigma}' \xleftarrow{\$} \mathsf{Sig.Rerand}(\sigma')$; $\tilde{\sigma}'' \xleftarrow{\$} \mathsf{Sig.Rerand}(\sigma'')$.	$\xrightarrow{C', D'', \mathsf{ZKPK}_6}$	
	$\xleftarrow{P', \mathsf{ZKPK}_7}$	Transform input: $P' \leftarrow \mathrm{e}(C', D'')^{k_\varrho^{-1}}$.
Unblind P': $P \leftarrow P'^{k_c^{-1} k_d^{-1}}$.		
Return $K' \leftarrow \frac{\hat{C} \prod_{i=0}^n \mathrm{e}(C_{i,1}, D_{i,1})}{P\, \mathrm{e}(C_0, D_0) \prod_{i=1}^n \mathrm{e}(C_{i,L_i,2}, D_{i,2})}$.		**Return** ε.

Fig. 5. The Query protocol. The proofs ZKPK_6 and ZKPK_7 are defined in the text.

key. Finally, the issuer signs the value $D_{0,2}$ using the Abe et al. signature scheme, such that the user can prove to the database that he uses the correct input in the Query protocol.

Decryption of a Record. Assume a user wants to decrypt a record; $\mathsf{HP\text{-}ABE}$ decryption would work as in Equation 4 assuming the user's key satisfies the ciphertext policy (cf. [NYO08]).

$$K' \leftarrow \frac{\hat{C} \prod_{i=0}^n \mathrm{e}(C_{i,1}, D_{i,1})}{\mathrm{e}(C_0, D_0) \prod_{i=0}^n \mathrm{e}(C_{i,L_i,2}, D_{i,2})} \tag{4}$$

Intuitively, each key component $D_{i,2}$ allows the user to decrypt a ciphertext component $C_{i,L_i,2}$ if $D_{i,2}$ corresponds to attribute L_i. However, in our scheme, even if the attributes $(L_1, \ldots L_n)$ in the user's key all satisfy the policy, he could by construction not decrypt the ciphertext component $C_{0,\varrho,2}$, because all users lack the key component for this zeroth attribute. Indeed, the key for this would be $D_{0,2}^{k_\varrho^{-1}}$, where k_ϱ is the database's secret. Thus, to decrypt, the user has to run the Query protocol with the database to compute $P \overset{\text{def}}{=} \mathrm{e}(C_{0,\varrho,2}, D_{0,2}^{k_\varrho^{-1}}) = \mathrm{e}(C_{0,\varrho,2}, D_{0,2})^{k_\varrho^{-1}}$ and then run the $\mathsf{HP\text{-}ABE}$ decryption. This protocol is given in Figure 5. It is of course important that 1) the database cannot learn $C_{0,\varrho,2}$ nor $D_{0,2}$ because otherwise it would learn which record a user attempts to decrypt and 2) the database is nevertheless ensured that the user indeed wants to compute this expression on valid inputs. The latter is achieved by having the user prove that he knows signatures on $C_{0,\varrho,2}$ and $D_{0,2}$ from the database and the issuer, and the former is ensured by proper blinding of these two inputs. Note that if the user does not possess the necessary decryption keys, decryption will result in a random symmetric key K' to be recovered (which will make the authenticated decryption algorithm $\mathsf{AuthDec}$ return $\perp$).

The user proves possession of signatures on $C_{0,\varrho,2}$ and $D_{0,2}$, specific to the record ψ and user φ respectively, by the proof protocol $\mathsf{ZKPK}_6 \overset{\text{def}}{=} \mathsf{ZKPK}\{(k_c^{-1}, k_d^{-1}, \tilde{\sigma}', \tilde{\sigma}'') : \mathsf{Sig.Prove}(vk_{\mathcal{DB}_\varrho}, C'^{k_c^{-1}}, \tilde{\sigma}') \wedge \mathsf{Sig.Prove}(vk_{\mathcal{I}}, D''^{k_d^{-1}}, \tilde{\sigma}'')\}$. By the proof protocol $\mathsf{ZKPK}_7 \overset{\text{def}}{=} \mathsf{ZKPK}\{(k_\varrho) : (P')^{k_\varrho} = \mathrm{e}(C', D'') \wedge A_{0,\varrho} = A_{0,0}^{k_\varrho}\}$ the database proves to the user that P' is indeed correctly computed with respect to its secret key k_ϱ that is defined by $A_{0,\varrho}$.

Real-world escrow. The issuer can recover the plaintext from a given ciphertext C with the help of $sk_{\mathcal{I}}$: $K' \leftarrow \hat{C}\,\mathrm{e}(C_0, g_2)^{-w\beta^{-1}} = K g_{\mathrm{T}}^{wr}\,\mathrm{e}(g_1^{\beta r}, g_2)^{-w\beta^{-1}} = K$. The issuer now checks if the authenticated ciphertext can be decrypted with $\mathsf{AuthDec}$ with key K. If $\mathsf{AuthDec}$ succeeded, the issuer can recover the ciphertext policy thus: $\{W_i \leftarrow \{t \in \mathbb{N}_{n_i} | C_{i,1}^{a_{i,t}} = C_{i,t,2}\}\}_{i=1}^{n}$. For $i = 0$, the perfect soundness of the NIZK_3 proof guarantees that $C_{0,\varrho,2}$ is well-formed and that $r = \sum r_i$, therefore $W_0 \leftarrow \{\varrho\}$. If $\mathsf{AuthDec}$ failed, the issuer simply sets $W_1 = \emptyset$ (the message can never be decrypted).

5.2 Protocol Complexity

A significant amount of work is done during setup and record issuing phases. Let N_{users}, N_{records}, and V denote the number of users, records, and attributes respectively. The players have to perform computations in time linear either to $N_{\text{users}} \cdot V$, or to $N_{\text{records}} \cdot V$, but never linear to the product of the three $N_{\text{users}} \cdot N_{\text{records}} \cdot V$, similarly to [CDNZ11].

For each query, the user has to do work linear in n (number of categories), while the database has to do only a constant[1] amount of work (in contrast, in [CDNZ11] both the user and the database have to do work linear in n). In the worst case, each user will query each ciphertext, so it is very important to ensure a fast query protocol for the database, which has to bear most of the work.

Adding revocation does not have a big influence on the complexity analysis: the complexity of the query phase remains the same. The public key of the issuer however will increase linearly in size to N_{users}.

6 Security Analysis

Theorem 1 *If the* $\mathsf{HP\text{-}ABE}$ *scheme by Nishide et al. is secure, the SXDH assumption holds, and the* $\max(N_{users}, N_{records})$*-SFP assumption holds in the chosen bilinear group, then our scheme presented in Section 5.1 securely implements the* HACOT *functionality described in Section 2.*

Corollary 2 *Our* HACOT *scheme is secure in the generic bilinear group model.*

The corollary trivially follows from security of the Nishide et al. scheme in the generic group model [Nis08]. We prove the theorem by demonstrating indistinguishability between adversarial actions in the real protocol and the ideal world for static corruptions: for every real-world adversary $\mathcal{A}$, we show how to construct an ideal-world adversary $\mathcal{S}$, such that for every environment $\mathcal{E}$: $\mathcal{E}$ cannot distinguish whether it is interacting with $\mathcal{A}$ in the real world or with $\mathcal{S}$ in the ideal world, i.e.,

$$\begin{aligned} &\forall \mathcal{A} : \exists \mathcal{S} : \forall \mathcal{E} : \mathrm{Adv}_{\mathcal{E},\mathcal{A}}^{\mathsf{HACOT}} \overset{\text{def}}{=} \\ &\left| \Pr\left[\mathrm{Exec}(\mathcal{E}, \mathcal{A}, \mathcal{R}_{\text{HACOT}}) \overset{\$?}{=} 1\right] - \Pr\left[\mathrm{Exec}(\mathcal{E}, \mathcal{S}, \mathcal{F}_{\text{HACOT}}) \overset{\$?}{=} 1\right] \right| = \textit{negl.} \ , \end{aligned} \tag{5}$$

where $\mathrm{Exec}(\mathcal{E}, \mathcal{A}, \mathcal{R}_{\text{HACOT}})$ denotes the binary random variable given by the output of $\mathcal{E}$ when interacting with $\mathcal{A}$ and $\mathcal{R}_{\text{HACOT}}$ in the real world; and analogously for $\mathrm{Exec}(\mathcal{E}, \mathcal{S}, \mathcal{F}_{\text{HACOT}})$ in the ideal world.

Proof sketch. Due to space constraints we only sketch the proof here; details are given in Appendix D. In case all parties are honest, the real and ideal worlds are indistinguishable by construction. To handle the case where some parties misbehave, we need to show how to construct a simulator $\mathcal{S}$. The construction of $\mathcal{S}$ is mostly straightforward: $\mathcal{S}$ extracts the witnesses from all interactive zero-knowledge proofs so that $\mathcal{S}$ can extract blinding factors and signature forgeries from $\mathcal{A}$ in the query protocol. $\mathcal{S}$ also simulates all interactive zero-knowledge proofs and GS proofs, which gives it the “wiggle room” to encrypt bogus plaintexts instead of the real records, and later manipulate the query protocol so that honest users still recover the correct record. We show that dishonest users cannot detect the deception based on the security of the $\mathsf{HP\text{-}ABE}$ scheme plus the SXDH assumption, where the latter is needed because of our tweaks in the “zeroth category” of the $\mathsf{HP\text{-}ABE}$ scheme. Another delicate point in the proof is that a dishonest issuer should not be able to issue to an honest user a malformed key that permits correct decryption even if access should be denied. We prove that the joint randomness in the issuance protocol prevents this under the SXDH assumption.

[1] Constant assuming we fix the security parameter κ. If κ varies, then the run time is $\mathrm{O}(\kappa^3)$.

7 Implementation

We implemented the scheme presented in Section 5 in C++ using the PBC Library [Lyn07]. Our implementation uses “D”-type curves [Lyn07], on which the SDXH assumption is believed to hold. Keys and records are stored as files on disk. Records can be files of arbitrary size. We use the NIST standard AES-256-CGM as the authenticated encryption scheme.

Our program currently does not support revocation.

We have observed run times of around 0.5–3 seconds for most algorithms/protocols when run with groups of order $\approx 2^{158}$ and 5 categories and 22 attributes, except the $\mathsf{IssueUserKey}$ protocol, which took around 11.5 seconds. Overall, the measured run times confirm our theoretical predictions.

8 Comparison with HACOT by Camenisch et al. at PKC 2011

In this section, we will compare the scheme presented by Camenisch et al. at PKC 2011 [CDNZ11] (CDNZ) with our scheme (without revocation). We first show that the class of expressible policies in both schemes is the same, and that there is an efficient transformation from CDNZ-policies to our policies (the transformation from our policies to CDNZ-polices is inefficient, as it may require an exponential number of categories. We omit it due to space constraints). We then compare the communications and runtime costs of both schemes in “steady state” (after setup) when they are used with CDNZ-policies, and show that our scheme is faster and needs less bandwidth.

Explanation of CDNZ-policies. In CDNZ, the keys issued to users are associated with a list of bits of length ℓ: $\boldsymbol{d} = (d_0, \dots, d_{\ell-1})$ and $d_i \in \mathbb{N}_2$. Each record in the database has an access control policy associated with it. A policy $\boldsymbol{c} = (c_0, \dots, c_{\ell-1})$ is also expressed as a list of ℓ bits ($c_i \in \mathbb{N}_2$). The index of a bit in that list is called its “category”. A key is authorized to access a record if and only if $\sum_{i=0}^{\ell-1} c_i d_i = \sum_{i=0}^{\ell-1} c_i$.

Expressing CDNZ Policies in our Scheme. It is easy to emulate CDNZ-policies in our scheme: simply set the same number of categories ($n \leftarrow \ell$), and create two attributes per category $\forall i \in \mathbb{N}^*_{n+1} : n_i = 2$. Users’ keys are endowed with the following attributes, and policies are formed thus: $\forall i \in \mathbb{N}^*_{n+1} : L_i \leftarrow d_i$; if $c_i = 0 : W_i \leftarrow \{0, 1\}$; else $W_i \leftarrow \{1\}$. We will use this transformation as the basis for comparing the two schemes.

Efficiency comparison. Tables 1, 2, and Figure 12 (in Appendix E) show that the scheme presented in this paper has lower communication and computation costs than CDNZ in “steady state” (after setup). Assuming we use our scheme to express CDNZ-policies ($n = \ell$, $V = 2\ell$), our records are (asymptotically) 1.6 times smaller, the database can generate them (asympotically) 1.8 times faster, and users can check them (asymptotically) twice as fast. The communication costs in the query protocol are constant in our scheme, versus linear in CDNZ. The user’s computational costs during the query protocol are (asymptotically) five times lower in our scheme; and the database’s computational costs are constant-time in our scheme (versus linear in CDNZ).

Our scheme becomes faster and requires less bandwidth than CDNZ when there are three categories or more.

It must be said however, that our scheme trades more efficient “steady state” communication, storage, and computational costs with more expensive setup costs: for example, it takes about 18 seconds (for $n = 10$) to issue a user’s key in our scheme, while this operation is nearly instantaneous in CDNZ.

9 Conclusions and Future Work

We created a scheme that allows a database to publish records that are protected by a hidden access control policy, and that users can access without revealing their identity or choice of record. Extensions to our scheme allow the key issuer to revoke the user’s keys. We have proved our scheme secure in the generic bilinear group model.

Our construction uses attribute-based encryption and, in comparison to the prior work based on anonymous credentials, offers more expressive policies and improved efficiency. Finally, we have implemented our scheme.

Timing results of a prototype implementation show that the scheme is scalable and sufficiently performant to be used in practical settings.

Recently, Okamoto and Takashima [OT12] have proposed an HP-ABE scheme that is secure under the decision linear assumption. It seems that one could extend their scheme into a HACOT scheme similarly as we have done it for the Nishide el al. scheme, i.e., by splitting the attribute keys between the database and the users. As this approach does not use the HP-ABE scheme as a block box, the security of such an extension of the Okamoto and Takashima scheme will have to be proven from scratch. We leave this as future work.

Other future work is investigating how to remove the unfettered access the issuer has over the published records.

Acknowledgements. The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) as part of the "ICT Trust and Security Research", under grant agreements n°257782 for the project ABC4Trust and n°216676 for the project Ecrypt II.

References

[AFG$^+$10] Masayuki Abe, Georg Fuchsbauer, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Structure-Preserving Signatures and Commitments to Group Elements. In Tal Rabin, editor, *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 209–236. Springer, 2010.

[AHO10] Masayuki Abe, Kristiyan Haralambiev, and Miyako Ohkubo. Signing on Elements in Bilinear Groups for Modular Protocol Design. *IACR Cryptology ePrint Archive*, 2010:133, 2010.

[ASM06] Man Au, Willy Susilo, and Yi Mu. Constant-size dynamic k-taa. In Roberto De Prisco and Moti Yung, editors, *Security and Cryptography for Networks*, volume 4116 of *Lecture Notes in Computer Science*, pages 111–125. Springer Berlin / Heidelberg, 2006.

[BB08] Dan Boneh and Xavier Boyen. Short signatures without random oracles and the sdh assumption in bilinear groups. *Journal of Cryptology*, 21:149–177, 2008.

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In Matt Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 227–242. Springer Berlin / Heidelberg, 2004.

[BN08] Mihir Bellare and Chanathip Namprempre. Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. *J. Cryptology*, 21(4):469–491, 2008.

[BSW07] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-Policy Attribute-Based Encryption. In *IEEE Symposium on Security and Privacy*, pages 321–334. IEEE Computer Society, 2007.

[Can00] Ran Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. *IACR Cryptology ePrint Archive*, 2000:67, 2000.

[CCS08] Jan Camenisch, Rafik Chaabouni, and Abhi Shelat. Efficient protocols for set membership and range proofs. In Josef Pieprzyk, editor, *Advances in Cryptology - ASIACRYPT 2008*, volume 5350 of *Lecture Notes in Computer Science*, pages 234–252. Springer Berlin / Heidelberg, 2008.

[CDM00] Ronald Cramer, Ivan Damgård, and Philip D. MacKenzie. Efficient Zero-Knowledge Proofs of Knowledge Without Intractability Assumptions. In Hideki Imai and Yuliang Zheng, editors, *Public Key Cryptography*, volume 1751 of *LNCS*, pages 354–373. Springer, 2000.

[CDN09] Jan Camenisch, Maria Dubovitskaya, and Gregory Neven. Oblivious Transfer with Access Control. In Ehab Al-Shaer, Somesh Jha, and Angelos D. Keromytis, editors, *ACM Conference on Computer and Communications Security*, pages 131–140. ACM, 2009.

[CDN10] Jan Camenisch, Maria Dubovitskaya, and Gregory Neven. Unlinkable Priced Oblivious Transfer with Rechargeable Wallets. In Radu Sion, editor, *Financial Cryptography*, volume 6052 of *LNCS*, pages 66–81. Springer, 2010.

[CDNZ11] Jan Camenisch, Maria Dubovitskaya, Gregory Neven, and Gregory M. Zaverucha. Oblivious Transfer with Hidden Access Control Policies. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *Public Key Cryptography*, volume 6571 of *LNCS*, pages 192–209. Springer, 2011.

[CGH09] Scott E. Coull, Matthew Green, and Susan Hohenberger. Controlling Access to an Oblivious Database Using Stateful Anonymous Credentials. In Stanislaw Jarecki and Gene Tsudik, editors, *Public Key Cryptography*, volume 5443 of *LNCS*, pages 501–520. Springer, 2009.

[CNS07] Jan Camenisch, Gregory Neven, and Abhi Shelat. Simulatable Adaptive Oblivious Transfer. In Moni Naor, editor, *EUROCRYPT*, volume 4515 of *LNCS*, pages 573–590. Springer, 2007.

[CS97] Jan Camenisch and Markus Stadler. Efficient Group Signature Schemes for Large Groups. In Burton S. Kaliski Jr., editor, *CRYPTO*, volume 1294 of *LNCS*, pages 410–424. Springer, 1997.

[GHW11] Matthew Green, Susan Hohenberger, and Brent Waters. Outsourcing the Decryption of ABE Ciphertexts. In *USENIX Security Symposium*. USENIX Association, 2011.

[GPS08] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for Cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, 2008.

[GS08] Jens Groth and Amit Sahai. Efficient Non-interactive Proof Systems for Bilinear Groups. In Nigel P. Smart, editor, *EUROCRYPT*, volume 4965 of *LNCS*, pages 415–432. Springer, 2008.

[GSW10] Essam Ghadafi, Nigel. Smart, and Bogdan Warinschi. Groth-Sahai proofs revisited. In Phong Nguyen and David Pointcheval, editors, *Public Key Cryptography – PKC 2010*, volume 6056 of *Lecture Notes in Computer Science*, pages 177–192. Springer Berlin / Heidelberg, 2010.

[KSW08] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate Encryption Supporting Disjunctions, Polynomial Equations, and Inner Products. In Nigel P. Smart, editor, *EUROCRYPT*, volume 4965 of *LNCS*, pages 146–162. Springer, 2008.

[LOS$^+$10] Allison B. Lewko, Tatsuaki Okamoto, Amit Sahai, Katsuyuki Takashima, and Brent Waters. Fully Secure Functional Encryption: Attribute-Based Encryption and (Hierarchical) Inner Product Encryption. In Henri Gilbert, editor, *EUROCRYPT*, volume 6110 of *LNCS*, pages 62–91. Springer, 2010.

[Lyn07] Ben Lynn. *On the Implementation of Pairing-Based Cryptography*. PhD thesis, Stanford University, 2007. PBC library available at `http://crypto.stanford.edu/pbc/`.

[NFHF09] Toru Nakanishi, Hiroki Fujii, Yuta Hira, and Nobuo Funabiki. Revocable group signature schemes with constant costs for signing and verifying. In Stanislaw Jarecki and Gene Tsudik, editors, *Public Key Cryptography – PKC 2009*, volume 5443 of *Lecture Notes in Computer Science*, pages 463–480. Springer Berlin / Heidelberg, 2009.

[NGSN10] Shivaramakrishnan Narayan, Martin Gagné, and Reihaneh Safavi-Naini. Privacy Preserving EHR System Using Attribute-based Infrastructure. In Adrian Perrig and Radu Sion, editors, *CCSW*, pages 47–52. ACM, 2010.

[Nis08] Takashi Nishide. *Cryptographic Schemes with Minimum Disclosure of Private Information in Attribute-Based Encryption and Multiparty Computation*. PhD thesis, University of Electro-Communications, 2008.

[NP99] Moni Naor and Benny Pinkas. Oblivious Transfer with Adaptive Queries. In Michael J. Wiener, editor, *CRYPTO*, volume 1666 of *LNCS*, pages 573–590. Springer, 1999.

[NYO08] Takashi Nishide, Kazuki Yoneyama, and Kazuo Ohta. Attribute-Based Encryption with Partially Hidden Encryptor-Specified Access Structures. In Steven M. Bellovin, Rosario Gennaro, Angelos D. Keromytis, and Moti Yung, editors, *ACNS*, volume 5037 of *LNCS*, pages 111–129, 2008.

[OT12] Tatsuaki Okamoto and Katsuyuki Takashima. Adaptively Attribute-Hiding (Hierarchical) Inner Product Encryption. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT*, volume 7237 of *LNCS*, pages 591–608. Springer, 2012.

[PW00] Birgit Pfitzmann and Michael Waidner. Composition and Integrity Preservation of Secure Reactive Systems. In Dimitris Gritzalis, Sushil Jajodia, and Pierangela Samarati, editors, *ACM Conference on Computer and Communications Security*, pages 245–254. ACM, 2000.

[PW01] Birgit Pfitzmann and Michael Waidner. A Model for Asynchronous Reactive Systems and its Application to Secure Message Transmission. In *IEEE Symposium on Security and Privacy*, pages 184–, 2001.

[Rab81] Michael O. Rabin. How to Exchange Secrets by Oblivious Transfer. Technical Report TR-81, Harvard Aiken Computation Laboratory, 1981.

[Sho04] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004. `http://eprint.iacr.org/`.

[SW05] Amit Sahai and Brent Waters. Fuzzy Identity-Based Encryption. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *LNCS*, pages 457–473. Springer, 2005.

[ZAW$^+$10] Ye Zhang, Man Ho Au, Duncan S. Wong, Qiong Huang, Nikos Mamoulis, David W. Cheung, and Siu-Ming Yiu. Oblivious Transfer with Access Control : Realizing Disjunction without Duplication. In Marc Joye, Atsuko Miyaji, and Akira Otsuka, editors, *Pairing*, volume 6487 of *LNCS*, pages 96–115. Springer, 2010.

A Preliminaries

A.1 Perfect Zero-knowledge Proofs of Knowledge

A zero-knowledge proof of knowledge (ZKPK) [CDM00] is a protocol in which a prover $\mathcal{P}$ can convince a verifier $\mathcal{Q}$ that he knows a secret — for instance, the discrete logarithm of a certain element, the discrete-logarithm–based representation of a certain element to certain bases, a pre-image of a bilinear pairing — without disclosing these to the verifier.

We will use the notation introduced by Camenisch and Stadler[CS97] when we need to perform a ZKPK in a protocol. For example:

$$\mathsf{ZKPK}\{(\alpha, \beta) : y = g^{\alpha} \wedge z = g^{\beta} h^{\alpha}\}$$

is used for proving the knowledge of the discrete logarithm of y to the base g, and a representation of z to the bases g and h such that the h-part of this representation is equal to the discrete logarithm of y to the base g [CS97]. We use the convention that letters in the first parenthesis denote the elements whose knowledge is proven, and all other letters denote elements which are known to the verifier.

With ZKPK protocols it is possible to work with statements of the form:

$$\mathsf{ZKPK}\Big\{\Big(\big(x_i \in \mathbb{Z}_p\big)_{i=0}^{u_x}, \big(\gamma_i \in \mathbb{G}_1\big)_{i=0}^{u_g}, \big(\eta_i \in \mathbb{G}_2\big)_{i=0}^{u_h}\Big) : \quad \bigwedge_{i=0}^{n_g} \Big(G_i = \prod_{j=0}^{n_{1,i}} g_{1,i,j}^{x_{\nu_{1,i,j}}}\Big) \tag{6}$$

$$\bigwedge_{i=0}^{n_h} \Big(H_i = \prod_{j=0}^{n_{2,i}} h_{2,i,j}^{x_{\nu_{2,i,j}}}\Big) \bigwedge_{i=0}^{n_t} \Big(T_i = \prod_{j=0}^{n_{3,i}} t_{3,i,j}^{x_{\nu_{3,i,j}}} \prod_{j=0}^{n_{4,i}} \mathrm{e}(g_{4,i,j}, \eta_{\nu_{4,i,j}}) \prod_{j=0}^{n_{5,i}} \mathrm{e}(\gamma_{\nu_{5,i,j}}, h_{5,i,j})\Big)\Big\}$$

where $t_{3,i,j}, T_i \in \mathbb{G}_\mathrm{T}$, $g_{1,i,j}, g_{4,i,j}, G_i \in \mathbb{G}_1$, $h_{2,i,j}, h_{5,i,j}, H_i \in \mathbb{G}_2$ and $\nu_{1,i,j}, \nu_{2,i,j}, \nu_{3,i,j} \in \mathbb{N}_{u_x+1}$, $\nu_{4,i,j} \in \mathbb{N}_{u_g+1}$, $\nu_{5,i,j} \in \mathbb{N}_{u_h+1}$ and $u_x, u_g, u_h, n_g, n_h, n_t, \big(n_{k,i}\big)_{k=1}^{5} \in \mathbb{N}$ are known to both parties.

A.2 Non-interactive Zero-knowledge Proofs

A non-interactive zero-knowledge (NIZK) proof allows a prover $\mathcal{P}$ to certify that a certain statement is satisfiable, without revealing a satisfiable assignment in the proof (however $\mathcal{P}$ usually needs to know such a satisfiable assignment to be able to compute the proof). The proof he issues can be checked offline by a verifier $\mathcal{Q}$.

Groth-Sahai Proofs In our construction, we will work with a subset of Groth-Sahai (GS) proofs [GS08,GSW10] which allows us to prove the satisfiability of the following classes of statements:

$$\mathsf{NIZK}\Big\{\Big(\{x_i \in \mathbb{Z}_p\}_{i=0}^{u}\Big) : \bigwedge_{i=0}^{n} \Big(G_i = \prod_{j=0}^{n_i} g_{i,j}^{x_{\nu_{i,j}}}\Big)\Big\}$$

where $G_i, g_{i,j} \in \mathbb{G}_1$ and $\nu_{i,j} \in \mathbb{N}_{u+1}$ and $u, n, n_i \in \mathbb{N}$ are known to both $\mathcal{P}$ and $\mathcal{Q}$. GS proofs require the SXDH assumption to hold.

GS proofs are not proofs of knowledge, since it is possible to combine several GS proofs to construct a GS proof of a related statement (as used in [CDNZ11]).

A.3 Structure-Preserving Signatures

A structure-preserving signature scheme is a tuple of algorithms $(\mathsf{Sig.KeyGen}, \mathsf{Sig.Sign}, \mathsf{Sig.Verify}, \mathsf{Sig.Rerand}, \mathsf{Sig.Prove}, \mathsf{Sig.KeyProve})$. The subset of the signature scheme by Abe et al. [AFG+10] shown below allows the message to be either an element of $\mathbb{G}_1$ or of $\mathbb{G}_2$ (it is also possible to sign a tuple of group elements, see [AFG+10]). We will show how the signature works for signing messages in $\mathbb{G}_2$. For messages in $\mathbb{G}_1$, simply switch $\mathbb{G}_1$ and $\mathbb{G}_2$ in everything below.

Figure 6 shows how to generate a signing and verification key pair, and Figure 7 shows how perform a proof of knowledge of the signing key. Figure 8 show how to sign a message $m \in \mathbb{G}_2$, and Figure 9 shows how to re-randomize a signature. Figure 10 show how to check a signature.

To prove possession of a signature σ for a message m without revealing σ nor m, one would proceed as follows: first, the signature is re-randomized: $\tilde{\sigma} \leftarrow \mathsf{Sig.Rerand}(\sigma)$; second the message is blinded: $k_m \overset{\$}{\leftarrow} \mathbb{Z}_p$; $\tilde{m} \leftarrow m^{k_m}$; and finally the proof of knowledge shown in Figure 11 is performed.

A.4 Authenticated Encryption

An authenticated encryption algorithm AuthEnc and corresponding decryption algorithm AuthDec is integrity-of-ciphertext secure [BN08] if no adversary $\mathcal{A}$ can, with non-negligible probability, when given access to an encryption oracle $\mathsf{AuthEnc}_K(\cdot)$ for a random key $K \overset{\$}{\leftarrow} \mathbb{G}_\mathrm{T}$, produce an authenticated ciphertext C_{auth} that was never output by the encryption oracle so that $\mathsf{AuthDec}_K(C_{auth}) \neq \bot$.

$$\begin{array}{l} g_R, f_U \xleftarrow{\$} \mathbb{G}_1^* \,;\ \gamma_Z, \delta_Z, \delta_M, \gamma_M, \alpha, \beta \xleftarrow{\$} \mathbb{Z}_p^* \,; \\ A \leftarrow \mathrm{e}(g_R, g_2^{\alpha}) \,;\ B \leftarrow \mathrm{e}(f_U, g_2^{\beta}) \,; \\ g_Z \leftarrow g_R^{\gamma_Z} \,;\ f_Z \leftarrow f_U^{\delta_Z} \,;\ f_M \leftarrow f_U^{\delta_M} \,;\ g_M \leftarrow g_R^{\gamma_M} \,; \\ sgk \leftarrow (\alpha, \beta, \gamma_Z, \delta_Z, \gamma_M, \delta_M) \,; \\ vk \leftarrow (g_Z, f_Z, g_R, f_U, g_M, f_M, A, B) \,; \\ \textbf{Output } (sgk, vk). \end{array}$$

Fig. 6. Key generation algorithm (Sig.KeyGen).

$$\begin{array}{l} \mathsf{Sig.KeyProve}(sgk, vk) \overset{\text{def}}{=} \\ \mathsf{ZKPK}\{(\gamma_Z, \delta_Z, \delta_M, \gamma_M, \alpha, \beta) : \\ A = \mathrm{e}(g_R, g_2)^{\alpha} \wedge B = \mathrm{e}(f_U, g_2)^{\beta} \wedge \\ g_Z = g_R^{\gamma_Z} \wedge f_Z = f_U^{\delta_Z} \wedge f_M = f_U^{\delta_M} \wedge g_M = g_R^{\gamma_M}\}. \end{array}$$

Fig. 7. Proof of knowledge of signing key.

$$\begin{array}{l} \text{To sign a message } m \in \mathbb{G}_2 : \\ \zeta, \rho, \tau, \varphi, \omega \xleftarrow{\$} \mathbb{Z}_p^* \,;\ Z \leftarrow g_2^{\zeta} \,; \\ R \leftarrow g_2^{\rho - \gamma_Z \zeta} m^{-\gamma_M} \,;\ S \leftarrow g_R^{\tau} \,;\ T \leftarrow g_2^{(\alpha - \rho)\tau^{-1}} \,; \\ U \leftarrow g_2^{\varphi - \delta_Z \zeta} m^{-\delta_M} \,;\ V \leftarrow f_U^{\omega} \,;\ W \leftarrow g_2^{(\beta - \varphi)\omega^{-1}} \,; \\ \textbf{Output } \sigma \leftarrow (Z, R, S, T, U, V, W). \end{array}$$

Fig. 8. Signing algorithm (Sig.Sign).

$$\begin{array}{l} \rho, \gamma, \tau, \omega \xleftarrow{\$} \mathbb{Z}_p^* \,; \\ \tilde{R} \leftarrow R T^{\rho} \,;\ \tilde{S} \leftarrow (S g_R^{-\rho})^{\gamma} \,;\ \tilde{T} \leftarrow T^{\gamma^{-1}} \,; \\ \tilde{U} \leftarrow U W^{\tau} \,;\ \tilde{V} \leftarrow (V f_U^{-\tau})^{\omega} \,;\ \tilde{W} \leftarrow W^{\omega^{-1}} \,; \\ \textbf{Output } \tilde{\sigma} \leftarrow (Z, \tilde{R}, \tilde{S}, \tilde{T}, \tilde{U}, \tilde{V}, \tilde{W}). \end{array}$$

Fig. 9. Re-randomization algorithm (Sig.Rerand).

$$\begin{array}{l} \text{Check that:} \\ A = \mathrm{e}(g_Z, Z)\, \mathrm{e}(g_R, R)\, \mathrm{e}(S, T)\, \mathrm{e}(g_M, m) \wedge \\ B = \mathrm{e}(f_Z, Z)\, \mathrm{e}(f_U, U)\, \mathrm{e}(V, W)\, \mathrm{e}(f_M, m). \end{array}$$

Fig. 10. Verification algorithm (Sig.Verify).

$$\begin{array}{l} \mathsf{Sig.Prove}(\tilde{\sigma}, \tilde{m}, k_m) \overset{\text{def}}{=} \mathsf{ZKPK}\{(k_m^{-1}, Z, \tilde{R}, \tilde{U}) : \\ A = \mathrm{e}(g_Z, Z)\, \mathrm{e}(g_R, \tilde{R})\, \mathrm{e}(\tilde{S}, \tilde{T})\, \mathrm{e}(g_M, \tilde{m})^{k_m^{-1}} \wedge \\ B = \mathrm{e}(f_Z, Z)\, \mathrm{e}(f_U, \tilde{U})\, \mathrm{e}(\tilde{V}, \tilde{W})\, \mathrm{e}(f_M, \tilde{m})^{k_m^{-1}}\}. \end{array}$$

Fig. 11. Proof of possession of a signature.

A.5 Security Game for Hidden–ciphertext-policy Attribute-based Encryption

We say that a key with list attributes L_φ is *admissible* for an HP-ABE challenge tuple (K_0, K_1, W_0, W_1) if:

- If $K_0 \neq K_1$, then $L_\varphi \not\models W_0 \wedge L_\varphi \not\models W_1$.
- Else if $K_0 = K_1$, then $(L_\varphi \not\models W_0 \wedge L_\varphi \not\models W_1) \vee (L_\varphi \models W_0 \wedge L_\varphi \models W_1)$

A HP-ABE scheme is (*non*-selectively) secure if no PPT adversary $\mathcal{A}$ has non-negligible advantage in the following game [Nis08][2]:

Setup. The challenger generates an issuer key and gives the public key to $\mathcal{A}$.

Phase 1. $\mathcal{A}$ submits a list of attributes $L_\varphi \in \mathcal{L}_L$ to the challenger. The latter then generates a user key with these attributes and gives it to $\mathcal{A}$. This phase may be repeated polynomially many times (with different φ).

Challenge. $\mathcal{A}$ submits two plaintexts K_0 and K_1 and two ciphertext policies $W_0 \in \mathcal{L}_W$ and $W_1 \in \mathcal{L}_W$ to the challenger, such that all the keys issued so far are admissible with respect to the challenge tuple (K_0, K_1, W_0, W_1). The challenger then flips a random bit b, runs the encryption algorithm with K_b as message and W_b as ciphertext policy, and gives $\mathcal{A}$ the resulting ciphertext.

Phase 2. Phase 1 is repeated, with the restriction that the key must be admissible. Like phase 1, $\mathcal{A}$ may ask for polynomially many admissible keys.

Guess. The adversary outputs a guess b' of b.

The advantage of $\mathcal{A}$ is defined as being $\left|\Pr[b' = b] - \frac{1}{2}\right|$ where the probability is taken over the random coins used by the challenger and the adversary.

[2] The paper [NYO08] defines only the *selective* security of an HP-ABE scheme. We included the full security definition here, since our security proof depends on the *non*-selective security game, which is present only in Nishide's PhD thesis [Nis08].

B Formal Ideal-World Definition

Messages exchanged between the players and $\mathcal{F}_{\mathrm{HACOT}}$ are formatted as $\langle \mathit{str}, \mathrm{arg1}, \mathrm{arg2}, \mathrm{arg3}\rangle$, where *str* is a string identifying the type of message, and arg1, arg2, arg3, ... are the payload.

Unless noted otherwise, all channels in the ideal world are authenticated, secure and integer.

(1 – Issuer setup) $\mathcal{F}_{\mathrm{HACOT}}$ waits for a message $\langle \mathit{init{:}i}, \mathcal{L}_L, \mathcal{L}_W\rangle$ from $\mathcal{I}$, fixing the set of attributes $\mathcal{L}_L$ keys can be endowed with, and the set of ciphertext policies $\mathcal{L}_W$. $\mathcal{F}_{\mathrm{HACOT}}$ relays the message to $\mathcal{S}$.
For each user $\mathcal{U}_\varphi$ in the system: $\mathcal{F}_{\mathrm{HACOT}}$ waits for a message $\langle \mathit{proof{:}i{:}i{:}}\varphi\rangle$ from $\mathcal{I}$, and relays it to $\mathcal{S}$. $\mathcal{F}_{\mathrm{HACOT}}$ waits for a message $\langle \mathit{proof{:}i{:}u{:}}\varphi\rangle$ from $\mathcal{U}_\varphi$, and relays it to $\mathcal{S}$. $\mathcal{F}_{\mathrm{HACOT}}$ waits for a message $\langle \mathit{proof{:}i{:}i{:}}\varphi\mathit{{:}ok}\rangle$ from $\mathcal{S}$, and relays it to $\mathcal{I}$. $\mathcal{F}_{\mathrm{HACOT}}$ waits for a message $\langle \mathit{proof{:}i{:}u{:}}\varphi\mathit{{:}ok}\rangle$ from $\mathcal{S}$, and relays it to $\mathcal{U}_\varphi$. For each database $\mathcal{DB}_\varrho$ in the system: $\mathcal{F}_{\mathrm{HACOT}}$ proceeds similarly (simply replace "*u*" with "*d*" and φ with ϱ in the above).
Note that after step (1) has completed, it is not possible to add users or databases.

(2 – Database setup) For every database $\mathcal{DB}_\varrho$ in the system: $\mathcal{F}_{\mathrm{HACOT}}$ waits for a message $\langle \mathit{init{:}d{:}}\varrho\rangle$ from $\mathcal{DB}_\varrho$. $\mathcal{F}_{\mathrm{HACOT}}$ then relays the message to $\mathcal{S}$.
For each user $\mathcal{U}_\varphi$ in the system: $\mathcal{F}_{\mathrm{HACOT}}$ waits for a message $\langle \mathit{proof{:}d{:}d{:}}\varrho{:}\varphi\rangle$ from $\mathcal{DB}_\varrho$, and relays it to $\mathcal{S}$. $\mathcal{F}_{\mathrm{HACOT}}$ waits for a message $\langle \mathit{proof{:}d{:}u{:}}\varrho{:}\varphi\rangle$ from $\mathcal{U}_\varphi$, and relays it to $\mathcal{S}$. $\mathcal{F}_{\mathrm{HACOT}}$ waits for a message $\langle \mathit{proof{:}d{:}d{:}}\varrho{:}\varphi\mathit{{:}ok}\rangle$ from $\mathcal{S}$, and relays it to $\mathcal{DB}_\varrho$. $\mathcal{F}_{\mathrm{HACOT}}$ waits for a message $\langle \mathit{proof{:}d{:}u{:}}\varrho{:}\varphi\mathit{{:}ok}\rangle$ from $\mathcal{S}$, and relays it to $\mathcal{U}_\varphi$. $\mathcal{F}_{\mathrm{HACOT}}$ proceeds similarly for the issuer (simply replace "*u*" with "*i*" remove φ in the above).

The steps (3) to (5) may now be run in any order, any number of times. It means that in the real world, records may be added to the database at any point of time without having to re-encrypt the whole database. The database provider just distributes an update containing the ciphertexts of the new records. The scheme of [CDNZ11] can easily be transformed to achieve this functionality as well.

(3 – Record issuance) Upon receiving $\langle \mathit{rec{:}}\varrho{:}\psi, K_{\varrho,\psi}, W_{\varrho,\psi}\rangle$ from $\mathcal{DB}_\varrho$, where $\mathcal{F}_{\mathrm{HACOT}}$ has not yet stored an entry ϱ, ψ for K; $K_{\varrho,\psi} \in \mathbb{G}_{\mathrm{T}}$ and $W_{\varrho,\psi} \in \mathcal{L}_W$; $\mathcal{F}_{\mathrm{HACOT}}$ stores $K_{\varrho,\psi}$ and $W_{\varrho,\psi}$ and sends $\langle \mathit{rec{:}}\varrho{:}\psi, \ell\,(K_{\varrho,\psi}), \ell\,(W_{\varrho,\psi})\rangle$ to $\mathcal{S}$. (ψ is the record identifier, and ℓ is a leakage function.) $\mathcal{F}_{\mathrm{HACOT}}$ waits for $\langle \mathit{rec{:}}\varrho{:}\psi\mathit{{:}ok}\rangle$ from $\mathcal{S}$. $\mathcal{F}_{\mathrm{HACOT}}$ then broadcasts $\langle \mathit{rec{:}}\varrho{:}\psi\mathit{{:}done}\rangle$ to all users. $\mathcal{F}_{\mathrm{HACOT}}$ also sends $\langle \mathit{rec{:}}\varrho{:}\psi\mathit{{:}escrow}, K_{\varrho,\psi}, W_{\varrho,\psi}\rangle$ to $\mathcal{I}$.

(4 – User key issuance) Upon receiving $\langle \mathit{key{:}u{:}}\varphi, L_\varphi\rangle$ from $\mathcal{U}_\varphi$, where $\mathcal{F}_{\mathrm{HACOT}}$ has not yet stored an entry φ for L; $L_\varphi \in \mathcal{L}_L$; $\mathcal{F}_{\mathrm{HACOT}}$ relays the message to $\mathcal{S}$. $\mathcal{F}_{\mathrm{HACOT}}$ now waits for $\langle \mathit{key{:}i{:}}\varphi, L_\varphi\rangle$ from $\mathcal{I}$, and relays it to $\mathcal{S}$. $\mathcal{F}_{\mathrm{HACOT}}$ now waits for $\langle \mathit{key{:}i{:}}\varphi\mathit{{:}ok}\rangle$ from $\mathcal{S}$, and relays it to $\mathcal{I}$. $\mathcal{F}_{\mathrm{HACOT}}$ now waits for $\langle \mathit{key{:}u{:}}\varphi\mathit{{:}ok}\rangle$ from $\mathcal{S}$, and relays it to $\mathcal{U}_\varphi$. Finally, $\mathcal{F}_{\mathrm{HACOT}}$ stores L_φ.

(5 – Query) $\mathcal{F}_{\mathrm{HACOT}}$ prepares a counter $\alpha{\leftarrow}0$ for this step. Upon receiving $\langle \mathit{query{:}u{:}}\varphi{:}\varrho{:}\psi\rangle$ from $\mathcal{U}_\varphi$, where $\mathcal{F}_{\mathrm{HACOT}}$ has stored L_φ and $K_{\varrho,\psi}$, $\mathcal{F}_{\mathrm{HACOT}}$ increments α and sends $\langle \mathit{query{:}u{:}}\varrho{:}\alpha\rangle$ to $\mathcal{S}$. $\mathcal{F}_{\mathrm{HACOT}}$ waits for $\langle \mathit{query{:}d{:}}\varrho{:}\alpha, b\rangle$ from $\mathcal{DB}_\varrho$, where b is a bit, and relays the message to $\mathcal{S}$. $\mathcal{F}_{\mathrm{HACOT}}$ now waits for $\langle \mathit{query{:}d{:}}\varrho{:}\alpha\mathit{{:}ok}\rangle$ from $\mathcal{S}$ and relays it to $\mathcal{DB}_\varrho$. If it is the first time that $\mathcal{U}_\varphi$ queried for the record ψ, $\mathcal{F}_{\mathrm{HACOT}}$ selects $K'_{\varrho,\psi,\varphi} \overset{\$}{\leftarrow} \mathbb{G}_{\mathrm{T}}$. $\mathcal{F}_{\mathrm{HACOT}}$ now waits for $\langle \mathit{query{:}u{:}}\varrho{:}\alpha\mathit{{:}ok}\rangle$ from $\mathcal{S}$ and sends: (a) if $b = 0$: $\langle \mathit{query{:}u{:}}\varphi{:}\varrho{:}\psi, b, \bot\rangle$; (b) else if $b = 1$ and $L_\varphi \models W_{\varrho,\psi}$: $\langle \mathit{query{:}u{:}}\varphi{:}\varrho{:}\psi, b, K_{\varrho,\psi}\rangle$; (c) else if $b = 1$ and $L_\varphi \not\models W_{\varrho,\psi}$: $\langle \mathit{query{:}u{:}}\varphi{:}\varrho{:}\psi, b, K'_{\varrho,\psi,\varphi}\rangle$.[3]

C Revocation of users

In this section, we show how to add a privacy-friendly mechanism to revoke users to our HACOT scheme. We will first provide a high-level overview of the revocation mechanism. We then update the definition of the HACOT scheme and adapt the security definition. Next, we describe additional building blocks needed for the construction. And finally, we present the construction.

C.1 High-level Idea

The issuer generates a key revocation list (KRL) each time the set of revoked users changes, and publishes it. In the query phase, users then have to prove to the database in zero-knowledge that their key has not been revoked.

[3] Note that authenticated encryption would detect that case and output $\bot$. In [CDNZ11], the corresponding step in the ideal world actually returned $\bot$, but this was a mistake: in the real world users have no way of deciding if the result of the Query step was correct or not.

To implement the KRL concept in our protocol we take a revocation technique introduced by Nakanishi et al. [NFHF09] and used in a recent paper by Camenisch et al. [CDNZ11]. In the later, the revoked users' identities are sorted in lexicographical order and the revocation list has a version number $krlid$. The revocation list contains signatures for the tuples ($krlid$, *start*$_i$, *end*$_i$) for each pair of neighboring revoked user identities (*start*$_i$, *end*$_i$). The interval between two neighboring revoked user identities is called a *non-revocation interval*.

During the setup phase, the issuer signs all possible distances within a non-revocation interval and makes them part of his public key. Each time the issuer wants to revoke a user's key, he adds it to the set of revoked users' IDs and creates a fresh public KRL by increasing $krlid$, sorting the updated set of the revoked users lexicographically and signing all the pairs of the neighboring revoked users' IDs.

When querying the database, the user takes the most recent KRL_{krlid} as additional input, and the database takes the version number $krlid$ of the most recent KRL as additional input. To show that his key is not revoked, the user proves in zero-knowledge that his user ID lies within a non-revocation interval by proving possession of a signature for a tuple ($krlid$, *start*$_i$, *end*$_i$), and valid signatures on the distances between his ID and the edges of this non-revocation interval. To improve efficiency and reduce the size of the issuer's public key, it is possible to revoke a number of dummy user identities by default to reduce the size of the non-revocation intervals.

C.2 Modification to the Definition of HACOT

To make KRL deployment possible, the interfaces of the IssuerSetup algorithm and Query protocol of Section 2.3 must be modified, and new algorithms GenerateKRL and CheckKRL are introduced.

- IssuerSetup takes an additional parameter $N_{krlsize}$, an estimate of the total number of users in the system. An additional output is the initial key revocation list KRL_0.
- GenerateKRL$(pk_{\mathcal{I}}, sk_{\mathcal{I}}, F) \xrightarrow{\$} KRL_{krlid}$.

The issuer runs this algorithm to revoke all users in the subset F. The issuer needs to provide his public and secret keys. The output of the algorithm is a key revocation list KRL_{krlid}, where $krlid$ is a counter that is incremented each time this function is called.

- CheckKRL$(pk_{\mathcal{I}}, KRL_{krlid}) \xrightarrow{\$} b$.

Upon receiving a new KRL from the issuer, each user and database runs this algorithm to test whether it is correctly formed or not. The output is a bit b indicating the result of this check.

- In the Query protocol, the user takes the most recent KRL_{krlid} as additional input, and the database takes the serial number $krlid$ of the most recent KRL as additional input. The protocol fails if the user and the database have a different version of the KRL, or if the user is on the list of revoked users in the KRL. Internally, the user performs a zero-knowledge set-membership proof [CCS08] for the database that he is in the set of non-revoked users.

C.3 Modification to the Security Definition

Ideal World. At any time after issuer and database setup: When $\mathcal{F}_{\text{HACOT}}$ receives a message $\langle$*krl:krlid*$, F\rangle$ from $\mathcal{I}$ (where $krlid$ is the serial number of the KRL, and F is a subset of revoked users), $\mathcal{F}_{\text{HACOT}}$ relays this message to $\mathcal{S}$. $\mathcal{F}_{\text{HACOT}}$ now waits for $\langle$*krl:krlid:ok*$, F\rangle$ from $\mathcal{S}$, and relays the message to all users and all databases.

In the Query phase, $\mathcal{F}_{\text{HACOT}}$ only accepts messages from users which are not in F.

Upon receiving $\langle\langle$*newkrl:krlid*$, F, KRL_{krlid}\rangle\rangle$ from $\mathcal{I}$, each database and user run the CheckKRL algorithm. If the algorithm returns $b = 1$, then they send $\langle\langle$*krl:krlid:ok*$, F\rangle\rangle$ to $\mathcal{E}$.

Discussion of Security Properties. Informally, the modified ideal world provides the following additional security properties:

Database Security Cheating users who have been revoked cannot successfully engage in a query protocol with an honest database (assuming the honest database has received a copy of the latest KRL).

User Security If a user engages successfully in a query protocol, the only information the database can deduce is that the user's key has not been revoked. The identity, and choice of record of that user remain private.

C.4 Additional Building Blocks

For efficiency reasons, we employ two additional signature schemes: Boneh-Boyen signatures [BB08] (BB), which allows us to sign a message consisting of one element of $\mathbb{Z}_p$, and BBS+ signatures [ASM06,BBS04] (BBS), which allows us to sign a message consisting of several elements of $\mathbb{Z}_p$. Both schemes support key generation (BB.KeyGen, BBS.KeyGen), signing (BB.Sign, BBS.Sign), verification (BB.Verify, BBS.Verify), proof of knowledge of the private key (BB.KeyProve, BBS.KeyProve), and zero-knowledge proof-of-possession of a signature (BB.Prove, BBS.Prove). Please refer to the cited publications for implementation details. Note that BB.Prove and BBS.Prove are also written out in PK_I of [CDNZ11].

Additionally, we make use of the variant of the structure-preserving signature Sig which can sign messages consisting of *two* elements of $\mathbb{G}_2$ (see [AFG$^+$10] for details). We do not give the details of the implementation of the 2-element Sig.Prove, since it follows immediately from the verification equations (6) and (7) of [AFG$^+$10] using a similar trick as we used for the 1-element Sig.Prove.

C.5 Our Construction with Revocation

In this subsection, we describe the construction of our HACOT scheme with revocation. We need to modify the IssuerSetup, VerifyIssuerKey, IssueUserKey and Query algorithms and protocols presented in Section 5; and add two new algorithms: GenerateKRL and CheckKRL.

System parameters. The CRS is augmented with two random elements of $\mathbb{G}_1^*$, which are needed for the BBS.Prove algorithm during the Query protocol.

Issuer Key Generation. The issuer uses the 2-element variant of the structure preserving signature scheme when calling Sig.KeyGen. Additionally, the issuer generates a different signing key pair $(sgkctr, vkctr) \xleftarrow{\$}$ $\mathsf{BB.KeyGen}(1^\kappa)$ for signing messages consisting of one element of $\mathbb{Z}_p$. The issuer then signs the integers 1 through $N_{krlsize}$: $\{\varsigma_i \leftarrow \mathsf{BB.Sign}_{sgkctr}(i)\}_{i=1}^{N_{krlsize}}$. Finally, the issuer generate another signing key pair $(sgkrl, vkrl) \xleftarrow{\$}$ $\mathsf{BBS.KeyGen}(1^\kappa)$ for signing messages consisting of three elements of $\mathbb{Z}_p$.

The issuer can now generate a zeroth KRL with the GenerateKRL algorithm, as described later.

Issuer Key Verification When checking the issuer key, the signatures $\{\varsigma_i\}_{i=1}^{N_{krlsize}}$ and the zeroth KRL have to be verified. The issuer has to additionally prove knowledge of the verification keys $sgkctr$ and $sgkrl$.

Issuing Decryption Keys to Users. The signature σ'' is computed on the key component $D_{0,2}$ and the user ID φ (user IDs must be unique and must be between 1 and $\mathrm{O}(N_{krlsize})$) : $\sigma'' \xleftarrow{\$} \mathsf{Sig.Sign}_{sgk_{\mathcal{I}}}((D_{0,2}, g_2^{\varphi}))$.

KRL Generation. To generate a new revocation list, the issuer starts by incrementing the counter $krlid$. He generates a set of intervals of valid keys $]start_i, end_i[$ for $i = 1..N_{\text{intervals}}$ (where $N_{\text{intevals}} \in \mathbb{N}$, $start_i, end_i \in \mathbb{Z}_p$), where the union of all intervals $\bigcup_{i=1}^{N_{\text{intervals}}}]start_i, end_i[$ contains all non-revoked users IDs φ, and contains no revoked user ID. Furthermore, we require than $1 \leq end_i - start_i \leq N_{krlsize} + 1$.

For each of these intervals, the issuer generate a BBS+ signature on the endpoints of that interval and on $krlid$: $\{\hat{\varsigma}_i \leftarrow \mathsf{BBS.Sign}_{sgkrl}((krlid, start_i, end_i))\}_{i=1}^{N_{\text{intervals}}}$

The KRL consists of the list of intervals and their signature: $KRL_{krlid} \leftarrow \big(krlid, \big(start_i, end_i, \hat{\varsigma}_i\big)_{i=1}^{N_{\text{intervals}}}\big)$.

KRL Verification. Upon receiving a fresh KRL, the user and database need to check all the signatures in the KRL, and the diameter of the non-revocation intervals: $\bigwedge_{i=0}^{N_{\text{intervals}}} (\mathsf{BBS.Verify}(vkrl, (krlid, start_i, end_i), \hat{\varsigma}_i) \wedge 1 \leq end_i - start_i \leq N_{krlsize} + 1)$.

Decryption of a Record. In the query phase, the user needs to find the interval j in the most recent KRL KRL^{krlid} containing his user ID φ: $\mathit{start}_j < \varphi < \mathit{end}_j$. The user can find such an interval if and only if his key has not been revoked. Let $\mathit{left} \overset{\text{def}}{=} \varphi - \mathit{start}_j$ and $\mathit{right} \overset{\text{def}}{=} \mathit{end}_j - \varphi$. As in the regular protocol, the user blinds the key and ciphertext component: $C' \leftarrow C_{0,\varrho,2}^{k_c}$ and $D'' \leftarrow D_{0,2}^{k_d}$; and re-randomizes the associated signatures: $\tilde{\sigma}' \overset{\$}{\leftarrow} \mathsf{Sig.Rerand}(\sigma')$ and $\tilde{\sigma}'' \overset{\$}{\leftarrow} \mathsf{Sig.Rerand}(\sigma'')$.

The proof protocol ZKPK_6 is replaced by $\mathsf{ZKPK}_{6^v} \overset{\text{def}}{=} \mathsf{ZKPK}\{(k_c^{-1}, k_d^{-1}, \varphi, \mathit{start}_j, \mathit{end}_j, \tilde{\sigma}', \tilde{\sigma}'', \hat{\varsigma}_j, \varsigma_{\mathit{left}}, \varsigma_{\mathit{right}}) : \mathsf{Sig.Prove}(vk_{\mathcal{DB}_\varrho}, C'^{k_c^{-1}}, \tilde{\sigma}') \wedge \mathsf{Sig.Prove}(vk_{\mathcal{I}}, (D''^{k_d^{-1}}, g_2^{\varphi}), \tilde{\sigma}'') \wedge \mathsf{BBS.Prove}(vkrl, (krlid, \mathit{start}_j, \mathit{end}_j), \hat{\varsigma}_j) \wedge \mathsf{BB.Prove}(vkctr, \varphi - \mathit{start}_j, \varsigma_{\mathit{left}}) \wedge \mathsf{BB.Prove}(vkctr, \mathit{end}_j - \varphi, \varsigma_{\mathit{right}})\}$. In $\mathsf{ZKPK}_{6^v}$the user proves possession of the following signatures: 1) the signature on the ciphertext component; 2) the signature on the key component and user ID; 3) the jth signature in the KRL; 4) the signature on the value *left*; 5) the signature on the value *right*. Signatures 1 and 2 serve a similar role as in ZKPK_6. Signatures 3, 4 and 5 ensure that the user ID lies within one of the non-revocation intervals. All the proofs of possession are anonymous, i.e., the user ID, the record index, and the chosen non-revocation interval are not leaked.

D Security analysis

This section is devoted to the proof of Theorem 1 presented in Section 6.

D.1 Informal introduction to the proof

Roughly speaking, we prove that any set of cooperating corrupted (dishonest) parties can essentially do no more harm than they could in an ideal version of the HACOT protocol, $\mathcal{F}_{\text{HACOT}}$ (the *ideal world*, which we formally defined in Section B).

To this end, we introduce an adversary which is composed of two parts: the *environment* $\mathcal{E}$, which tells the honest parties in the system what they should do over the real world interface $\mathcal{R}_{\text{HACOT}}$ and the *real-world adversary* $\mathcal{A}$ which takes control of the corrupted parties and may behave arbitrarily. $\mathcal{E}$ and $\mathcal{A}$ may send arbitrary messages to each other. (One can show that it is sufficient to have $\mathcal{A}$ be a dummy: $\mathcal{A}$ relays all the messages he gets from $\mathcal{E}$ to the real world protocol, and relays all messages he gets from the protocol to $\mathcal{E}$ [Can00].)

The core idea of the proof is to show how to construct a simulator $\mathcal{S}$ of the "real-world adversary" which runs with the ideal scheme $\mathcal{F}_{\text{HACOT}}$ instead of the real one. When $\mathcal{E}$ is run with the ideal HACOT scheme, $\mathcal{E}$ tells the honest parties what they should do through the ideal world interface $\mathcal{F}_{\text{HACOT}}$ (which is identical to the real world interface), and where $\mathcal{A}$ has been replaced by $\mathcal{S}$; $\mathcal{E}$ will not be able to tell the difference.

(Please see section 6 for the formal statement of the theorem.)

D.2 Organization of the proof

We will consider the two cases where the issuer is honest and where the issuer is dishonest separately. For each case, we must deal with honest and dishonest databases, as well as honest and dishonest users. We will start by showing how to construct the simulator $\mathcal{S}$. Later, we will prove that the view of $\mathcal{E}$ in the real and ideal world are computationally indistinguishable. The proofs of the supporting lemmas have been moved to their own subsections at the end of this Section, as to not interrupt the flow of the proof with technicalities. We conclude this section by a discussion of the limitations of our proof.

D.3 Construction of the simulator

In the HACOT scheme, there is one issuer (who might be honest or dishonest), a certain number $\mathrm{N}_{\mathcal{DB}}$ of databases of which $\mathrm{N}_{\text{corr-}\mathcal{DB}}$ are corrupted ($0 \leq \mathrm{N}_{\text{corr-}\mathcal{DB}} \leq \mathrm{N}_{\mathcal{DB}}$), and a certain number $\mathrm{N}_{\mathcal{U}}$ of users of which $\mathrm{N}_{\text{corr-}\mathcal{U}}$ are corrupted ($0 \leq \mathrm{N}_{\text{corr-}\mathcal{U}} \leq \mathrm{N}_{\mathcal{U}}$).

The simulator $\mathcal{S}$ internally runs a copy of $\mathcal{A}$. $\mathcal{S}$ relays all the messages he receives from $\mathcal{E}$ to his internal copy of $\mathcal{A}$, and all relays all outbound messages from $\mathcal{A}$ to $\mathcal{E}$. Based on the behaviour of $\mathcal{A}$, $\mathcal{S}$ will play the role of all the corrupt parties in the ideal world setting. For the sake of $\mathcal{A}$, $\mathcal{S}$ simulates the behaviour of each of the ideal honest parties based on the information he receives from $\mathcal{F}_{\mathrm{HACOT}}$.

Like in the real world, we assume that $\mathcal{S}$ simply ignores malformed messages from $\mathcal{A}$. Also, $\mathcal{S}$ will ignore the last message of an interactive proof from $\mathcal{A}$ if it does not satisfy the ZKPK verification step.

In what follows, we will surround by quotes (for example "$\mathcal{I}$") all the corrupted parties controlled by $\mathcal{A}$ and all the honest parties internally simulated by $\mathcal{S}$ (which all operate in the "real world", i.e., these parties run the cryptographic algorithms and protocols). We will not put quotes around (for example $\mathcal{I}$) all honest ideal parties, and all corrupted ideal parties assumed by $\mathcal{S}$.

The numbers in the following two subsections correspond to the numbering of the real and ideal world definitions (see Section B).

Honest issuer Before the start of the protocol, $\mathcal{S}$ chooses a *hiding* common reference string for the Groth-Sahai proofs (NIZK_3).

(1 – Issuer setup) Upon receiving $\langle \textit{init:i}, \mathcal{L}_L, \mathcal{L}_W\rangle$ from $\mathcal{F}_{\mathrm{HACOT}}$, $\mathcal{S}$ runs $\mathsf{IssuerSetup}$ on behalf of "$\mathcal{I}$" and broadcasts $\langle\!\langle \textit{init:i:pk}, pk_{\mathcal{I}}\rangle\!\rangle$.
Upon receiving $\langle \textit{proof:i:i:}\varphi\rangle$ from $\mathcal{F}_{\mathrm{HACOT}}$ (where $\mathcal{U}_\varphi$ is an honest ideal user), $\mathcal{S}$ waits for the matching $\langle \textit{proof:i:u:}\varphi\rangle$ from $\mathcal{F}_{\mathrm{HACOT}}$, and sends both $\langle \textit{proof:i:i:}\varphi\textit{:ok}\rangle$ and $\langle \textit{proof:i:u:}\varphi\textit{:ok}\rangle$ to $\mathcal{F}_{\mathrm{HACOT}}$.
Upon receiving $\langle \textit{proof:i:i:}\varphi\rangle$ from $\mathcal{F}_{\mathrm{HACOT}}$ (where "$\mathcal{U}_\varphi$" is controlled by $\mathcal{A}$), $\mathcal{S}$ runs the $\mathsf{VerifyIssuerKey}$ protocol on behalf of "issuer" with "$\mathcal{U}_\varphi$". During the protocol, $\mathcal{S}$ simulates ZKPK_1. If the protocol completed successfully, $\mathcal{S}$ sends out $\langle \textit{proof:i:u:}\varphi\rangle$, $\langle \textit{proof:i:i:}\varphi\textit{:ok}\rangle$ and $\langle \textit{proof:i:u:}\varphi\textit{:ok}\rangle$ to $\mathcal{F}_{\mathrm{HACOT}}$.
($\mathcal{S}$ behaves in a similar way for the $\langle \textit{proof:i:d:}\varrho\rangle$ messages.)

(2 – Database setup) **(honest database)** Upon receiving $\langle \textit{init:d:}\varrho\rangle$ from $\mathcal{F}_{\mathrm{HACOT}}$ ($\mathcal{DB}_\varrho$ is therefore an honest ideal database), $\mathcal{S}$ runs the $\mathsf{DBSetup}$ algorithm on behalf of "$\mathcal{DB}_\varrho$" and broadcasts $\langle\!\langle \textit{init:d:pk}, pk_{\mathcal{DB}_\varrho}\rangle\!\rangle$.
Upon receiving $\langle \textit{proof:d:d:}\varrho\textit{:}\varphi\rangle$ from $\mathcal{F}_{\mathrm{HACOT}}$ (where $\mathcal{U}_\varphi$ is an honest ideal user), $\mathcal{S}$ waits for the matching $\langle \textit{proof:d:u:}\varrho\textit{:}\varphi\rangle$ from $\mathcal{F}_{\mathrm{HACOT}}$, and sends both $\langle \textit{proof:d:d:}\varrho\textit{:}\varphi\textit{:ok}\rangle$ and $\langle \textit{proof:d:u:}\varrho\textit{:}\varphi\textit{:ok}\rangle$ to $\mathcal{F}_{\mathrm{HACOT}}$.
Upon receiving $\langle \textit{proof:d:d:}\varphi\rangle$ from $\mathcal{F}_{\mathrm{HACOT}}$ (where "$\mathcal{U}_\varphi$ " is controlled by $\mathcal{A}$), $\mathcal{S}$ runs the $\mathsf{VerifyDBKey}$ protocol on behalf of "$\mathcal{DB}_\varrho$" with "$\mathcal{U}_\varphi$". During the protocol, $\mathcal{S}$ simulates ZKPK_2. If the protocol completed successfully, $\mathcal{S}$ sends out $\langle \textit{proof:d:u:}\varphi\rangle$, $\langle \textit{proof:d:d:}\varphi\textit{:ok}\rangle$ and $\langle \textit{proof:d:u:}\varphi\textit{:ok}\rangle$ to $\mathcal{F}_{\mathrm{HACOT}}$.
($\mathcal{S}$ behaves in a similar way for the $\langle \textit{proof:d:i:}\varrho\rangle$ messages.)

(dishonest database) Upon receiving $\langle\!\langle \textit{init:d:pk}, pk_{\mathcal{DB}_\varrho}\rangle\!\rangle$ from "$\mathcal{DB}_\varrho$" (controlled by $\mathcal{A}$), $\mathcal{S}$ sends $\langle \textit{init:d:}\varrho\rangle$ to $\mathcal{F}_{\mathrm{HACOT}}$. When "$\mathcal{DB}_\varrho$" wants to runs the $\mathsf{VerifyDBKey}$ protocol with "$\mathcal{U}_\varphi$" (where "$\mathcal{U}_\varphi$" is controlled by $\mathcal{S}$), $\mathcal{S}$ sends $\langle \textit{proof:d:d:}\varrho\textit{:}\varphi\rangle$ to $\mathcal{F}_{\mathrm{HACOT}}$. $\mathcal{S}$ waits until it receives $\langle \textit{proof:d:u:}\varrho\textit{:}\varphi\rangle$ from $\mathcal{F}_{\mathrm{HACOT}}$, before letting "$\mathcal{U}_\varphi$" run his side of the protocol. During the protocol, $\mathcal{S}$ extracts the database secret key $sk_{\mathcal{DB}_\varrho}$ in ZKPK_2. If the protocol completed successfully, $\mathcal{S}$ sends out $\langle \textit{proof:d:d:}\varphi\textit{:ok}\rangle$ and $\langle \textit{proof:d:u:}\varphi\textit{:ok}\rangle$ to $\mathcal{F}_{\mathrm{HACOT}}$.
($\mathcal{S}$ behaves in a similar way for the $\langle \textit{proof:d:i:}\varrho\rangle$ messages.)

(3 – Record issuance) **(honest database)** Upon receiving $\langle \textit{rec:}\varrho\textit{:}\psi, \ell\,(K_{\varrho,\psi}), \ell\,(W_{\varrho,\psi})\rangle$ from $\mathcal{F}_{\mathrm{HACOT}}$, $\mathcal{S}$ chooses a *random* policy $\widetilde{W}_{\varrho,\psi} \xleftarrow{\$} \mathcal{L}_W$ and a *random* plaintext $\widetilde{K}_{\varrho,\psi} \xleftarrow{\$} \mathbb{G}_{\mathrm{T}}$, and runs $\mathsf{IssueRecord}$ on behalf of "$\mathcal{DB}_\varrho$", yielding the bogus ciphertext $\widetilde{C}^{\varrho,\psi}$. $\mathcal{S}$ simulates NIZK_3 during the algorithm. $\mathcal{S}$ then broadcasts $\langle\!\langle \textit{rec:}\varrho\textit{:}\psi\textit{:ct}, C_{\varrho,\psi}\rangle\!\rangle$ broadcasts to every user controlled by $\mathcal{A}$. Then $\mathcal{S}$ sends $\langle \textit{rec:}\varrho\textit{:}\psi\textit{:ok}\rangle$ back to $\mathcal{F}_{\mathrm{HACOT}}$. (Note that with high probability $\widetilde{W}_{\varrho,\psi} \neq W_{\varrho,\psi}$, and $\widetilde{K}_{\varrho,\psi} \neq K_{\varrho,\psi}$.)

(dishonest database) Upon receiving $\langle\!\langle \textit{rec:}\varrho\textit{:}\psi\textit{:ct}, C_{\varrho,\psi}\rangle\!\rangle$ from "$\mathcal{DB}_\varrho$" (controlled by $\mathcal{A}$), $\mathcal{S}$ runs $\mathsf{CheckRecord}$. If the check failed, $\mathcal{S}$ ignores the message. Else $\mathcal{S}$ runs Escrow on behalf of "$\mathcal{I}$", recovering the plaintext $K_{\varrho,\psi}$ and the policy $W_{\varrho,\psi}$ of the record. $\mathcal{S}$ then sends $\langle \textit{rec:}\varrho\textit{:}\psi, K_{\varrho,\psi}, W_{\varrho,\psi}\rangle$ to $\mathcal{F}_{\mathrm{HACOT}}$. Once $\mathcal{S}$ has received this message on behalf of all the users it controls and the issuer, $\mathcal{S}$ sends $\langle \textit{rec:}\varrho\textit{:}\psi\textit{:ok}\rangle$ back to $\mathcal{F}_{\mathrm{HACOT}}$.

(4 - User key issuance) **(honest user)** Upon receiving $\langle \textit{key:u:}\varphi, L_\varphi\rangle$ from $\mathcal{F}_{\mathrm{HACOT}}$, $\mathcal{S}$ waits for a message $\langle \textit{key:i:}\varphi, L_\varphi, b\rangle$ from $\mathcal{F}_{\mathrm{HACOT}}$. If $b = 1$, the $\mathcal{S}$ runs $\mathsf{IssueUserKey}$ internally on behalf of "$\mathcal{U}_\varphi$" and "$\mathcal{I}$". Then $\mathcal{S}$ sends $\langle \textit{key:i:}\varphi\textit{:ok}\rangle$ and $\langle \textit{key:u:}\varphi\textit{:ok}\rangle$ back to $\mathcal{F}_{\mathrm{HACOT}}$.

(dishonest user) Upon receiving $\langle key{:}i{:}\varphi, L_\varphi, b\rangle$ from $\mathcal{F}_{\text{HACOT}}$ where "$\mathcal{U}_\varphi$" is controlled by $\mathcal{A}$, run IssueUserKey on behalf of "$\mathcal{I}$" with "$\mathcal{U}_\varphi$". If $b = 0$, then $\mathcal{S}$ aborts the protocol before the last message. If $b = 1$, then $\mathcal{S}$ chooses the key $sk_{\mathcal{U}\varphi}$ of $\mathcal{U}_\varphi$ independently from the input of $\mathcal{U}_\varphi$ ($\mathcal{S}$ has to extract the witnesses from ZKPK_4 and simulate ZKPK_5 to be able to do that). After that, $\mathcal{S}$ sends $\langle key{:}u{:}\varphi, L_\varphi\rangle$, $\langle key{:}i{:}\varphi{:}ok\rangle$ and $\langle key{:}u{:}\varphi{:}ok\rangle$ to $\mathcal{F}_{\text{HACOT}}$.

(5 – Query) **(honest user, honest database)** Upon receiving $\langle query{:}u{:}\varrho{:}\alpha\rangle$ from $\mathcal{F}_{\text{HACOT}}$, where $\mathcal{DB}_\varrho$ is honest, $\mathcal{S}$ waits for a message $\langle query{:}d{:}\varrho{:}\alpha, b\rangle$ from $\mathcal{F}_{\text{HACOT}}$. $\mathcal{S}$ then sends $\langle query{:}d{:}\varrho{:}\alpha{:}ok\rangle$ and $\langle query{:}u{:}\varrho{:}\alpha{:}ok\rangle$ back to $\mathcal{F}_{\text{HACOT}}$.

(honest user, dishonest database) Upon receiving $\langle query{:}u{:}\varrho{:}\alpha\rangle$ from $\mathcal{F}_{\text{HACOT}}$, where "$\mathcal{DB}_\varrho$" is controlled by $\mathcal{A}$, $\mathcal{S}$ chooses any of the users he controls, and any of the records that where published by "$\mathcal{DB}_\varrho$". $\mathcal{S}$ then runs the Query protocol with "$\mathcal{DB}_\varrho$" on behalf of the chosen user. During the protocol, $\mathcal{S}$ simulates ZKPK_6 and extracts the witnesses from ZKPK_7. If "$\mathcal{DB}_\varrho$" is cooperative in the query, $\mathcal{S}$ sets $b = 1$, else $b = 0$. $\mathcal{S}$ then sends $\langle query{:}d{:}\varrho{:}\alpha, b\rangle$, $\langle query{:}d{:}\varrho{:}\alpha{:}ok\rangle$ and $\langle query{:}u{:}\varrho{:}\alpha{:}ok\rangle$ back to $\mathcal{F}_{\text{HACOT}}$.

(dishonest user, honest database) When a dishonest user (controlled by the adversary) wants to perform a Query protocol with "$\mathcal{DB}_\varrho$" (controlled by simulator), $\mathcal{S}$ extracts k_c and k_d from ZKPK_6, which allows him to recover $D_{0,2}$ and $C_{0,\varrho,2}$, thus uniquely identifying the user "$\mathcal{U}_\varphi$" who makes the query, and the record used in that query (remember that $\mathcal{S}$ chose these two values without influence from $\mathcal{A}$). $\mathcal{S}$ now sends $\langle query{:}u{:}\varphi{:}\varrho{:}\psi\rangle$ to $\mathcal{F}_{\text{HACOT}}$, and waits for $\langle query{:}u{:}\varrho{:}\alpha\rangle$ from $\mathcal{F}_{\text{HACOT}}$. $\mathcal{S}$ now sends $\langle query{:}d{:}\varrho{:}\alpha{:}ok\rangle$ and $\langle query{:}u{:}\varrho{:}\alpha{:}ok\rangle$ to $\mathcal{F}_{\text{HACOT}}$, $\mathcal{S}$ now waits for $\langle query{:}u{:}\varphi{:}\varrho{:}\psi, b, K'\rangle$ from $\mathcal{F}_{\text{HACOT}}$, where $K' = K_{\varrho,\psi}$ or $K' = \perp$ or K' is random.

If $b = 0$, simulator aborts the query protocol before sending the last message. If $b = 1$, the simulator simulates the two-party computation so that $\mathcal{U}_\varphi$ recovers the same plaintext K' he would have recovered in the HP-ABE scheme, instead of the bogus plaintext ($\mathcal{S}$ must simulate ZKPK_7 to be able to do that).

(dishonest user, dishonest database) This case is handled internally by $\mathcal{A}$, and $\mathcal{S}$ doesn't need to do anything.

Dishonest issuer Before the start of the protocol, $\mathcal{S}$ chooses a *hiding* common reference string for the Groth-Sahai proofs (NIZK_3).

(1 – Issuer setup) Upon receiving $\langle\langle init{:}i{:}pk, pk_{\mathcal{I}}\rangle\rangle$ from "$\mathcal{I}$" (controlled by $\mathcal{A}$) on behalf of a database or a user, $\mathcal{S}$ sends $\langle init{:}i, \mathcal{L}_L, \mathcal{L}_W\rangle$ to $\mathcal{F}_{\text{HACOT}}$.

When "$\mathcal{I}$" wants to runs the VerifyIssuerKey protocol with "$\mathcal{U}_\varphi$" (where "$\mathcal{U}_\varphi$" is controlled by $\mathcal{S}$), $\mathcal{S}$ sends $\langle proof{:}i{:}i{:}\varphi\rangle$ to $\mathcal{F}_{\text{HACOT}}$. $\mathcal{S}$ waits until it receives $\langle proof{:}i{:}u{:}\varphi\rangle$ from $\mathcal{F}_{\text{HACOT}}$, before letting "$\mathcal{U}_\varphi$" run his side of the protocol. During the protocol, $\mathcal{S}$ extracts the issuer secret key $sk_{\mathcal{I}}$ in ZKPK_1. If the protocol completed successfully, $\mathcal{S}$ sends out $\langle proof{:}i{:}i{:}\varphi{:}ok\rangle$ and $\langle proof{:}i{:}u{:}\varphi{:}ok\rangle$ to $\mathcal{F}_{\text{HACOT}}$.

Since $\mathcal{S}$ will miss the initialization messages of dishonest users controlled by $\mathcal{A}$, $\mathcal{S}$ must make sure he controls at least one user $\mathcal{U}_\varphi$ in the ideal world. $\mathcal{S}$ sends $\langle proof{:}i{:}i{:}\varphi\rangle$, $\langle proof{:}i{:}u{:}\varphi\rangle$, $\langle proof{:}i{:}i{:}\varphi{:}ok\rangle$, $\langle proof{:}i{:}u{:}\varphi{:}ok\rangle$ to $\mathcal{F}_{\text{HACOT}}$ for that.

($\mathcal{S}$ proceeds in the same way for databases who want to check the issuer key.)

(If there are no honest users and no honest databases in the system, then $\mathcal{S}$ will miss this message. But in that case, the whole simulation is internal to $\mathcal{A}$ anyway.)

(2 – Database setup) **(honest database)** This case is the same as for the honest issuer case, except that VerifyDBKey is also done with "$\mathcal{I}$" (controlled by $\mathcal{A}$).

(dishonest database) This case is almost the same as for the honest issuer case. The difference is that if there are no honest users in the system, $\mathcal{S}$ will not receive this message. This is not a problem, as in that case all the dishonest databases are internal to $\mathcal{A}$.

(3 – Record issuance) **(honest database)** Upon receiving $\langle rec{:}\varrho{:}\psi, \ell(K_{\varrho,\psi}), \ell(W_{\varrho,\psi})\rangle$ from $\mathcal{F}_{\text{HACOT}}$, $\mathcal{S}$ sends $\langle rec{:}\varrho{:}\psi{:}ok\rangle$ back to $\mathcal{F}_{\text{HACOT}}$. $\mathcal{S}$ waits for $\langle rec{:}\varrho{:}\psi{:}escrow, K_{\varrho,\psi}, W_{\varrho,\psi}\rangle$ from $\mathcal{F}_{\text{HACOT}}$, and then runs IssueRecord on behalf of "$\mathcal{DB}_\varrho$", yielding $C_{\varrho,\psi}$. $\mathcal{S}$ then broadcasts $C_{\varrho,\psi}$ to all users controlled by $\mathcal{A}$.

(dishonest database) Upon receiving $\langle \mathit{rec}{:}\varrho{:}\psi{:}\mathit{ct}, C_{\varrho,\psi}\rangle$ from $\mathcal{DB}_\varrho$ (controlled by $\mathcal{A}$), $\mathcal{S}$ uses $sk_\mathcal{I}$ (which he got from step 1) to run the Escrow algorithm and recover $K_{\varrho,\psi}$ and $W_{\varrho,\psi}$. $\mathcal{S}$ then sends $\langle \mathit{rec}{:}\varrho{:}\psi, K_{\varrho,\psi}, W_{\varrho,\psi}\rangle$ and $\langle \mathit{rec}{:}\varrho{:}\psi{:}\mathit{ok}\rangle$ to $\mathcal{F}_{\text{HACOT}}$.

(4 – User key issuance) **(honest user)** Upon receiving $\langle \mathit{key}{:}u{:}\varphi, L_\varphi\rangle$ from $\mathcal{F}_{\text{HACOT}}$, $\mathcal{S}$ runs $\mathsf{IssueUserKey}$ with "$\mathcal{I}$" on behalf of "$\mathcal{U}_\varphi$", except that it simulates ZKPK_4 and extracts the witnesses from ZKPK_5. If "$\mathcal{I}$" was cooperative, the simulator sets $b = 1$, else $b = 0$. After that, $\mathcal{S}$ sends $\langle \mathit{key}{:}i{:}\varphi, L_\varphi, b\rangle$, $\langle \mathit{key}{:}i{:}\varphi{:}\mathit{ok}\rangle$, and $\langle \mathit{key}{:}u{:}\varphi{:}\mathit{ok}\rangle$ to $\mathcal{F}_{\text{HACOT}}$.

(dishonest user) This case is handled internally by $\mathcal{A}$, and $\mathcal{S}$ doesn't need to do anything.

(5 – Query) **(honest user, honest database)** This case is the same as for the honest issuer case.

(honest user, dishonest database) This case is the same as for the honest issuer case.

(dishonest user, honest database) This step is triggered when a dishonest user (controlled by the adversary) wants to perform a Query protocol with "$\mathcal{DB}_\varrho$" (controlled by simulator).

If $\mathcal{S}$ doesn't yet control any malicious user in the ideal world, he will need to create one user $\mathcal{U}_\varphi$ (this creation is invisible to $\mathcal{E}$): $\mathcal{S}$ chooses any random $L_\varphi \in \mathcal{L}_L$, and sends $\langle \mathit{key}{:}u{:}\varphi, L_\varphi\rangle$, $\langle \mathit{key}{:}i{:}\varphi, L_\varphi, b\rangle$, $\langle \mathit{key}{:}i{:}\varphi{:}\mathit{ok}\rangle$, and $\langle \mathit{key}{:}u{:}\varphi{:}\mathit{ok}\rangle$ to $\mathcal{F}_{\text{HACOT}}$.

$\mathcal{S}$ now sends $\langle \mathit{query}{:}u{:}\varphi{:}\varrho{:}\psi\rangle$ to $\mathcal{F}_{\text{HACOT}}$, and waits for $\langle \mathit{query}{:}u{:}\varrho{:}\alpha\rangle$ from $\mathcal{F}_{\text{HACOT}}$. $\mathcal{S}$ now sends $\langle \mathit{query}{:}d{:}\varrho{:}\alpha{:}\mathit{ok}\rangle$ and $\langle \mathit{query}{:}u{:}\varrho{:}\alpha{:}\mathit{ok}\rangle$ to $\mathcal{F}_{\text{HACOT}}$, $\mathcal{S}$ now waits for $\langle \mathit{query}{:}u{:}\varphi{:}\varrho{:}\psi, b, K'\rangle$ from $\mathcal{F}_{\text{HACOT}}$, where $K' = K_{\varrho,\psi}$ or $K' = \bot$ or K' is random. If $b = 1$, simulator finishes the query protocol, except that it extracts from ZKPK_6 and simulates ZKPK_7. If $b = 0$, simulator aborts the query protocol before sending the last message.

(dishonest user, dishonest database) This case is handled internally by $\mathcal{A}$, and $\mathcal{S}$ doesn't need to do anything.

D.4 Proof of indistinguishability

We are going to define a sequence of games Game_1 to $\mathsf{Game}_{N_{\text{games}}}$, as described by Shoup [Sho04]. In the zeroth game, everything is distributed as in the real world, whereas in the last game everything is distributed as in the ideal world. By the piling-up lemma, the advantage of $\mathcal{E}$ is less than the sum of the advantages in distinguishing between Games i and Game $i+1$. We are going to prove that $\mathcal{E}$ only has negligible advantage in distinguishing between two consecutive games, based either on a reduction to a hard cryptographic problem, or by *failure events* happening with negligible probability. As long as the number of games is polynomial with respect to the security parameter, the total advantage of $\mathcal{E}$ is negligible.

We must stress that in all intermediate games, the simulator $\mathcal{S}_i$ receives the inputs of all honest parties (i.e., we are not in the ideal world yet). By the last game, $\mathcal{S}_{N_{\text{games}}}$ will have isolated the part that interacts with $\mathcal{A}$ (which is exactly $\mathcal{S}$) from the part that behaves like $\mathcal{F}_{\text{HACOT}}$. This last game is then indistinguishable from the ideal world.

Game_1*:* As observed in the previous paragraph, $\mathcal{S}_1$ receives the input of all honest parties. $\mathcal{S}_1$ simply runs the parties he controls like in the real world. By construction, this setting is perfectly indistinguishable from the real world.

Game_2*:* $\mathcal{S}_2$ runs like $\mathcal{S}_1$, except that each time $\mathcal{S}_2$ runs an interactive zero-knowledge proof with $\mathcal{A}$, he either simulates the proof (if $\mathcal{S}_2$ is the prover) or extracts the witnesses from the proof (if $\mathcal{S}_2$ is the verifier).

$\mathcal{S}_2$ aborts if this fails. The probability that this happens is equal to the knowledge error of the ZKPKs, which is negligible. The advantage that $\mathcal{E}$ has in distinguishing between Game_2 and Game_1 is therefore negligible.

Game_3*:* $\mathcal{S}_3$ runs like $\mathcal{S}_2$, except that $\mathcal{S}_3$ chooses a *hiding* CRS for the Groth-Sahai proofs NIZK_3.

$\mathcal{S}_3$ aborts if "$\mathcal{DB}_\varrho$" (controlled by $\mathcal{A}$) manages to produce an unsound NIZK_3, i.e., if $C_0 \neq \prod_{i=0}^{n} C_{i,1}^{\beta}$ or $C_{0,\varrho,2} \neq C_{0,0}^{a_{0,0} k_\varrho}$ (which would result in Escrow returning the wrong ciphertext policy). Because in the normal Groth-Sahai setting (*binding* CRS) we have perfect soundness, $\mathcal{A}$ (and by extension $\mathcal{E}$) can only do that if it successfully distinguished the *hiding* CRS from a *binding* one.

If $\mathcal{E}$ has a significant advantage in distinguishing between Game_3 and Game_2, then we can use $\mathcal{E}$ to win the SXDH game with the same advantage [GS08].

Game_4*:* $\mathcal{S}_4$ runs like $\mathcal{S}_3$, except that each time $\mathcal{S}_4$ computes NIZK_3, he simulates the proof.

Because of the strong composable witness indistinguishability property of Groth-Sahai proofs, the view of $\mathcal{E}$ in Game_4 and Game_3 is perfectly the same.

Game_5*:* $\mathcal{S}_5$ runs like $\mathcal{S}_4$, except that $\mathcal{S}_5$ aborts if a user "$\mathcal{U}_\varphi$" (controlled by $\mathcal{S}_5$) can decrypt C_{auth} using the $K_{\varrho,\psi}$ (recovered from Query) without error from from AuthDec when he should be unable to decrypt.

Lemma 1. *The probability that the failure event of* Game_5 *happens is negligible.*

See Section D.5 for the proof. The advantage that $\mathcal{E}$ has in distinguishing between Game_5 and Game_4 is therefore negligible.

Game_6*:* $\mathcal{S}_6$ runs like $\mathcal{S}_5$, except that when an honest ideal user performs the Query protocol with a dishonest database $\mathcal{DB}_\varrho$, $\mathcal{S}_6$ chooses a random user "$\mathcal{U}_\varphi$" among the ones he controls and performs the Query protocol with "$\mathcal{DB}_\varrho$" on behalf of "$\mathcal{U}_\varphi$".

The view of $\mathcal{E}$ in Game_6 and Game_5 is perfectly the same, thanks to the zero-knowledge property of ZKPK_6 and the fact that all elements sent to "$\mathcal{DB}_\varrho$" were perfectly blinded by "$\mathcal{U}_\varphi$".

Game_7*:* $\mathcal{S}_7$ runs like $\mathcal{S}_6$, except that when a dishonest user "$\mathcal{U}_\varphi$" runs IssueUserKey with "$\mathcal{I}$" (controlled by $\mathcal{S}_7$), $\mathcal{S}_7$ ignores the input of "$\mathcal{U}_\varphi$" when computing $sk_{\mathcal{U}\varphi}$ (ZKPK_5 is now the proof of a false statement).

The view of $\mathcal{E}$ in Game_7 and Game_6 is perfectly the same as $\mathcal{S}_7$ does not change the distribution of $sk_{\mathcal{U}\varphi}$, and due to the zero-knowledge property of ZKPK_5.

Game_8*:* $\mathcal{S}_8$ runs like $\mathcal{S}_7$, except that when a dishonest user "$\mathcal{U}_\varphi$" (controlled by $\mathcal{A}$) runs the Query protocol with an honest database "$\mathcal{DB}_\varrho$" (controlled by $\mathcal{S}_8$), $\mathcal{S}_8$ recovers k_c and k_d (which allows him to unblind C' and D''), as well as the complete signature on the unblinded values $C_{0,\varrho,2}$ and $D_{0,2}$. $\mathcal{S}_8$ aborts if he never issued the signature on $C_{0,\varrho,2}$.

If $\mathcal{S}_8$ controls "$\mathcal{I}$", and $\mathcal{S}_8$ never issued the signature on $D_{0,2}$, then $\mathcal{S}_8$ aborts.

The probability of $\mathcal{S}_8$ aborting in Game_8 is equal to the probability of $\mathcal{A}$ carrying out an existential forgery on the signature scheme. Since the signature scheme we use is existentially unforgeable under chosen message attacks, this probability is negligible. The advantage that $\mathcal{E}$ has in distinguishing between Game_8 and Game_7 is therefore negligible.

Game_9*:* $\mathcal{S}_9$ runs like $\mathcal{S}_8$, except that when a dishonest user "$\mathcal{U}_\varphi$" (controlled by $\mathcal{A}$) runs the Query protocol with an honest database "$\mathcal{DB}_\varrho$" (controlled by $\mathcal{S}_9$), and is denied access to the record ψ, and it is the first time that $\mathcal{U}_\varphi$ queries for ψ, $\mathcal{S}_9$ sends "$\mathcal{U}_\varphi$" a random value $P' \xleftarrow{\$} \mathbb{G}_\mathrm{T}$ in the last step of the Query protocol (ZKPK_7 is now the proof of a false statement).

On all subsequent queries, $\mathcal{S}_9$ sends the adjusted value $P'_{(new)} \leftarrow P'^{\left(\frac{k_c^{(new)} k_d^{(new)}}{k_c^{(old)} k_d^{(old)}}\right)}_{(old)}$, so that "$\mathcal{U}_\varphi$" recovers the same value of P after unblinding.

Lemma 2. *If the SXDH assumption holds and the* HP-ABE *scheme is secure, then the advantage of $\mathcal{E}$ in distinguishing between* Game_9 *and* Game_8 *is negligible.*

See Section D.6 for the proof.

Game_{10}*:* $\mathcal{S}_{10}$ runs like $\mathcal{S}_9$, except that when an honest ideal database $\mathcal{DB}_\varrho$ issues a record, $\mathcal{S}_{10}$ runs IssueRecord on behalf of "$\mathcal{DB}_\varrho$" with a random plaintext $\widetilde{K}_{\varrho,\psi} \xleftarrow{\$} \mathbb{G}_\mathrm{T}$ and a random policy $\widetilde{W}_{\varrho,\psi} \xleftarrow{\$} \mathcal{L}_W$ as input, instead of the plaintext and policy used by $\mathcal{DB}_\varrho$.

When a dishonest user "$\mathcal{U}_\varphi$" (controlled by $\mathcal{A}$) runs the Query protocol with "$\mathcal{DB}_\varrho$", $\mathcal{S}_{10}$ sends "$\mathcal{U}_\varphi$" the value $P' \leftarrow K'^{k_c k_d}$ in the last step of the Query protocol (ZKPK_7 is now the proof of a false statement); where $K' = K_{\varrho,\psi}$ if the user is granted access, else K' is uniformly random. "$\mathcal{U}_\varphi$" will recover the same plaintext K' after unblinding P' in both Game_{10} and Game_9.

Lemma 3. *If the* $\mathsf{HP\text{-}ABE}$ *scheme is secure, then the advantage of* $\mathcal{E}$ *in distinguishing between* Game_{10} *and* Game_{9} *is negligible.*

See Section D.7 for the proof.

Game_{11}*:* $\mathcal{S}_{11}$ runs a copy of $\mathcal{F}_{\text{HACOT}}$ internally, and runs a copy of $\mathcal{S}$ (as constructed in D.3) to interact between $\mathcal{A}$ and $\mathcal{F}_{\text{HACOT}}$.

By construction, the view of $\mathcal{E}$ in Game_{11} and Game_{10} is perfectly the same.

Conclusion: The view of $\mathcal{E}$ in Game_{11} and the ideal world is now perfectly the same. Therefore, if the $\mathsf{HP\text{-}ABE}$ scheme is secure, the SXDH assumption holds, the SFP assumption holds, and $\mathsf{AuthEnc}$ is integrity-of-ciphertext secure, then $\mathcal{E}$ has only a negligible advantage in distinguishing between the real world and the ideal world. This concludes the proof.

D.5 Proof of lemma 1

Remember that the decryption algorithm of the $\mathsf{HP\text{-}ABE}$ scheme is:

$$K' \leftarrow \frac{\hat{C}\prod_{i=1}^{n} \mathrm{e}(C_{i,1}, D_{i,1})}{\mathrm{e}(C_0, D_0)\prod_{i=1}^{n} \mathrm{e}(C_{i,L_i,2}, D_{i,2})} = K g_{\mathrm{T}}^{-\sum_{i=1}^{n} \lambda_i \epsilon_{i,L_i}}. \tag{7}$$

In what follows, we assume that $\tilde{K}'$ is a correct decryption key for C_{auth} in $\mathsf{AuthDec}$. (Note that we don't necessarily need $\tilde{K}' = K$: remember that in that case Escrow sets $W_{\varrho,\psi} = \emptyset$).

$\mathcal{S}_5$ aborts in Game_5 if for some $\tilde{K}'$:

$$\exists j, 0 \leq j \leq n : \quad \epsilon_{j,L_j} \neq 0 \pmod{p} \quad \wedge \quad K g_{\mathrm{T}}^{-\sum_{i=1}^{n} \lambda_i \epsilon_{i,L_i}} = \tilde{K}' \pmod{p}.$$

If "$\mathcal{I}$" or "$\mathcal{DB}_\varrho$" (or both) are honest, then $\sum_{i=1}^{n} \lambda_i \epsilon_{i,L_i}$ is random (the honest "$\mathcal{I}$" randomizes the λ's, the honest "$\mathcal{DB}_\varrho$" randomizes the ϵ's). The probability of $\mathcal{S}_5$ aborting in that case are negligible.

For the rest of the section, will handle the case where both "$\mathcal{DB}_\varrho$" and "$\mathcal{I}$" are malicious.

If there exists a PPT algorithm $\mathcal{A}$ which causes $\mathcal{S}_5$ to abort in Game_5 with non-negligible advantage, we show how to construct a PPT algorithm $\mathcal{S}(\mathcal{A})$ which can either break the semantic security of El-Gamal encryption (which would in turn break the SXDH assumption) with advantage $(N_{\text{users}} N_{\text{records}} N_{\text{categories}})^{-1}$ that of $\mathcal{A}$, which is still a non-negligible advantage, or break the integrity-of-ciphertext security of the authenticated encryption scheme.

$\mathcal{S}(\mathcal{A})$ receives an El-Gamal public key $\bar{X} = g_2^{\bar{x}}$ from the challenger of the El-Gamal semantic-security game. $\mathcal{S}(\mathcal{A})$ chooses $\bar{m}_0, \bar{m}_1 \xleftarrow{\$} \mathbb{Z}_p^*$. It computes $\bar{K}_0 = g_2^{\bar{m}_0}$ and $\bar{K}_1 = g_2^{\bar{m}_1}$ and gives them to the challenger. The challenger flips a coin $\bar{b}$ and gives $\mathcal{S}(\mathcal{A})$ the ciphertexts $\bar{C}_1 = \bar{K}_{\bar{b}} X^{\bar{r}}$, $\bar{C}_2 = g_2^{\bar{r}}$. $\mathcal{S}(\mathcal{A})$ proceeds as follows: it chooses a key ϕ at random from the set of honest users and a value $\bar{\jmath}$ at random. In the key issuing protocol for key ϕ it sends $\bar{X}$ as its El-Gamal public key. It sends $\bar{C}_1$ and $\bar{C}_2$ instead of $E_{\bar{\jmath}}^{(\phi)}$ and $F_{\bar{\jmath}}^{(\phi)}$. All other values of E_i and F_i can be computed normally (these are simply El-Gamal encryptions). $\mathcal{S}(\mathcal{A})$ then fakes ZKPK_4.

$\mathcal{S}(\mathcal{A})$ extracts the values λ_i' and $a_{i,t}$ from $\mathcal{A}$ in ZKPK_5. $\mathcal{S}(\mathcal{A})$ can therefore recover the values of $\lambda_i^{(\phi)} = \lambda_i' + \lambda_i''$ for all $i \neq \bar{\jmath}$ and, using the escrow functionality, can compute $g_1^{\epsilon_{i,t}}$ from the ciphertext components $C_{i,t,2}$.

With probability at least the advantage of $\mathcal{A}$ (which is a non-negligible probability): There will be a query in which a key φ that does not satisfy the ciphertext policy of a given record, but where the recovered plaintext after $\mathsf{HP\text{-}ABE}$ decryption $\tilde{K}'$ can be used to decrypt C_{auth} without error using $\mathsf{AuthDec}$. If $\varphi \neq \phi$, $\mathcal{S}(\mathcal{A})$ aborts and takes a random guess. If the ciphertext policy of record satisfies $L_{\bar{\jmath}}^{(\phi)} \in W_{\bar{\jmath}}$, then $\mathcal{S}(\mathcal{A})$ aborts. Otherwise:

By setting $\lambda_{\bar{\jmath}} = \bar{m}_b + \lambda_{\bar{\jmath}}'$ for $b = 0$ or $b = 1$, $\mathcal{S}(\mathcal{A})$ checks for which of $b = 0$ or $b = 1$, the key K' can be used to decrypt C_{auth} with $\mathsf{AuthDec}$ without an error message:

$$K' \leftarrow K \prod_{i=0}^{n} \left(g_1^{\epsilon_{i,L_i}} \right)^{\lambda_i}.$$

$\mathcal{S}(\mathcal{A})$ outputs the value of b for which the above test is true.

(The probability that the above equation is satisfied for both values of $\bar{m}_0$ and $\bar{m}_1$ is negligible, since the value $\bar{m}_{1-b}$ is random, and the issuer never saw this value. The authenticated ciphertext C_{auth} in the record's ciphertext must therefore have a non-negligible chance to decrypt correctly under a random key K'. Such a ciphertext immediately gives rise to an attack on the integrity-of-ciphertext security of the authenticated encryption scheme.)

D.6 Proof of lemma 2

We are going to define an intermediate game for the sake of this proof:

$\mathsf{Game}_{8.5}$ $\mathcal{S}_{8.5}$ runs like $\mathcal{S}_8$, except that when a dishonest user "$\mathcal{U}_\varphi$" (controlled by $\mathcal{A}$) runs the Query protocol with an honest database "$\mathcal{DB}_\varrho$" (controlled by $\mathcal{S}_{8.5}$) for a record ψ, $\mathcal{S}_{8.5}$ sends "$\mathcal{U}_\varphi$" an incorrect value $P' \xleftarrow{\$} \mathbb{G}_\mathrm{T}$ in the last step of the Query protocol (ZKPK_7 may now be the proof of a false statement):

$$P' \leftarrow \left(\frac{\hat{C}^{(\psi)} \prod_{i=0}^{n} \mathrm{e}(C_{i,1}^{(\psi)}, D_{i,1}^{(\varphi)})}{K^{(\psi)}\, \mathrm{e}(C_0^{(\psi)}, D_0^{(\varphi)}) \prod_{i=1}^{n} \mathrm{e}(Q_{i,L_i^{(\varphi)},2}^{(\psi)}, D_{i,2}^{(\varphi)})} \right)^{k_c k_d}, \tag{8}$$

$$\text{where } \forall i \in \mathbb{N}_{n+1}^{*} \forall t \in \mathbb{N}_{n_i} : Q_{i,t,2}^{(\psi)} \begin{cases} \leftarrow C_{i,t,2}^{(\psi)} & \text{if } t \in W_i^{(\psi)} \text{ (component in policy);} \\ \xleftarrow{\$} \mathbb{G}_1 & \text{if } t \notin W_i^{(\psi)} \text{ (component not in policy).} \end{cases}$$

On all subsequent queries for the same record ψ, $\mathcal{S}_{8.5}$ does *not* re-randomize the Q-values.

Note that here the values P' are not *uniformly* random, like in Game_9. Also note that the value of P' is actually correct for records which "$\mathcal{U}_\varphi$" is authorized to access.

We will now prove that $\mathcal{E}$ cannot distinguish between Game_8 and $\mathsf{Game}_{8.5}$ with significant advantage if the $\mathsf{HP\text{-}ABE}$ scheme is secure. Afterwards we will prove that $\mathcal{E}$ cannot distinguish between $\mathsf{Game}_{8.5}$ and Game_9 with significant advantage if the SXDH assumption holds.

Proof of indistinguishability between Game_8 *and* $\mathsf{Game}_{8.5}$. If there exists a PPT algorithm $\mathcal{D}$ which distinguishes between Game_8 and $\mathsf{Game}_{8.5}$ with non-negligible advantage, then we can construct a PPT algorithm $\mathcal{S}(\mathcal{D})$ which runs $\mathcal{D}$ internally and which plays the $\mathsf{HP\text{-}ABE}$ game with non-negligible advantage.

We defines the sequence of games: Game_{8-0} to $\mathsf{Game}_{8-N_\text{records}}$. In $\mathsf{Game}_{8-\omega}$, the value P' for all records $\psi \leq \omega$ is computed as in $\mathsf{Game}_{8.5}$, and the value P' for all records $\psi > \omega$ is computed as in Game_8. Clearly, Game_{8-0} is exactly Game_8, and $\mathsf{Game}_{8-N_\text{records}}$ is exactly $\mathsf{Game}_{8.5}$. The existence of $\mathcal{D}$ implies the existence of $\mathcal{D}_\omega$ (for some ω) which can distinguish between $\mathsf{Game}_{8-(\omega-1)}$ and $\mathsf{Game}_{8-\omega}$ with an advantage N_records^{-1} that of $\mathcal{D}$.

The challenger starts by generating the issuer key, which he transmits to $\mathcal{S}(\mathcal{D}_\omega)$. $\mathcal{S}(\mathcal{D}_\omega)$ relays it to $\mathcal{D}_\omega$.

$\mathcal{S}(\mathcal{D}_\omega)$ runs $\mathcal{D}_\omega$, who may request a key with attribute list $\{L_i\}$ where $L_0 = 0$. $\mathcal{S}(\mathcal{D}_\omega)$ relays the request to the challenger and the response back to $\mathcal{D}_\omega$. This can be repeated polynomially many times.

When $\mathcal{D}_\omega$ wants to encrypt a record $\psi \neq \omega$ (not the challenge record), with plaintext $K^{(\psi)}$ and policy $W^{(\psi)}$ (with $W_0^{(\psi)} = \{\varrho\}$), $\mathcal{S}(\mathcal{D}_\omega)$ computes the ciphertext under the policy $W^{(\psi)\prime} = \{\{0, \varrho\}, W_1^{(\psi)}, W_2^{(\psi)}, \ldots, W_n^{(\psi)}\}$ (which includes the "issuer" attribute), and sends every component except $C_{0,0,2}^{(\psi)}$ (the one corresponding to the "issuer" attribute) to $\mathcal{D}_\omega$.

When $\mathcal{D}_\omega$ wants to encrypt the challenge record $\psi = \omega$ with plaintext $K^{(\omega)}$ and policy $W^{(\omega)}$ (with $W_0^{(\omega)} = \{\varrho\}$), $\mathcal{S}(\mathcal{D}_\omega)$ generates the following challenge plaintexts and policies: $K^{(0)} \leftarrow K^{(\omega)}$ and $W^{(0)} \leftarrow \{\{\varrho\}, \mathbb{N}_{n_1}, \mathbb{N}_{n_2}, \ldots, \mathbb{N}_{n_n}\}$ (the most liberal policy), as well as $K^{(1)} \leftarrow K^{(\omega)}$ and $W^{(1)} \leftarrow W^{(\omega)}$ (these are acceptable, since all keys issued will satisfy neither of $W^{(0)}$ and $W^{(1)}$). These are then sent to the challenger. The challenger flips a bit $b \xleftarrow{\$} \mathbb{N}_2$ and encrypts $K^{(b)}$ under policy $W^{(b)}$ yielding the ciphertext:

$$\{\tilde{C}^{(b)}, C_0^{(b)}, C_{0,1}^{(b)}, \{C_{0,0,2}^{(b)}, C_{0,\varrho,2}^{(b)}\}, \{C_{i,1}^{(b)}, \{\underline{Q_{i,t,2}^{(b)}}\}_{t=0}^{n_i-1}\}_{i=1}^{n}\}.$$

$\mathcal{S}(\mathcal{D}_\omega)$ now computes $C^{(b)}_{i,t,2}$ as follows:

$$\begin{aligned} &\forall i \in \mathbb{N}^*_{n+1} : \forall t \in W^{(\omega)}_i : && C^{(b)}_{i,t,2} \leftarrow Q^{(b)}_{i,t,2}; \\ &\forall i \in \mathbb{N}^*_{n+1} : \forall t \in \mathbb{N}_{n_i} \setminus W^{(\omega)}_i : && C^{(b)}_{i,t,2} \xleftarrow{\$} \mathbb{G}_1. \end{aligned}$$

That is, $\mathcal{S}(\mathcal{D}_\omega)$ re-randomizes all ciphertext components that are not in the policy. $\mathcal{S}(\mathcal{D}_\omega)$ now sends the following modified ciphertext to $\mathcal{D}_\omega$:

$$\{\tilde{C}^{(b)}, C^{(b)}_0, C^{(b)}_{0,1}, \{C^{(b)}_{0,\varrho,2}\}, \{C^{(b)}_{i,1}, \{\underline{C^{(b)}_{i,t,2}}\}_{t=0}^{n_i-1}\}_{i=1}^{n}\}.$$

In the view of $\mathcal{D}_\omega$, this challenge ciphertext is distributed exactly as if it was encrypted under the ciphertext policy $W^{(\omega)}$.

$\mathcal{S}(\mathcal{D}_\omega)$ runs $\mathcal{D}_\omega$, who may now ask for more keys (polynomially many times).

To process queries by $\mathcal{D}_\omega$, $\mathcal{S}(\mathcal{D}_\omega)$ first extracts from ZKPK_6 to recover the key index φ and the record index ψ. $\mathcal{S}(\mathcal{D}_\omega)$ will have no problem in processing queries for the non-challenge records $\psi \neq \omega$ to return P' of the requisite distribution. For the challenge record $\psi = \omega$, $\mathcal{S}(\mathcal{D}_\omega)$ computes P' as follows, and sends the result back to $\mathcal{D}_\omega$:

$$P' \leftarrow \left(\frac{\hat{C}^{(b)} \prod_{i=0}^{n} \mathrm{e}(C^{(b)}_{i,1}, D^{(\varphi)}_{i,1})}{K^{(0)}\, \mathrm{e}(C^{(b)}_0, D^{(\varphi)}_0) \prod_{i=1}^{n} \mathrm{e}(Q^{(b)}_{i,L^{(\varphi)}_i,2}, D^{(\varphi)}_{i,2})} \right)^{k_c k_d} . \tag{9}$$

If $b = 0$ or if the key φ satisfies the policy of record ω, then all values of $Q_{i,t,2}$ in the above equation are non-random, we therefore have that $P' = \mathrm{e}(C^{(\omega)}_{0,\varrho,2}, D^{(\varphi)}_{0,2})^{k_\varrho^{-1} k_c k_d}$ and thus distributed exactly as in Game_8. Else P' is distributed exactly in $\mathsf{Game}_{8.5}$ (it contains random values that $\mathcal{D}_\omega$ doesn't have).

The query phase may be repeated polynomially many times.

$\mathcal{S}(\mathcal{D}_\omega)$ runs $\mathcal{D}_\omega$, who now finally outputs a guess of b. $\mathcal{S}(\mathcal{D}_\omega)$ sends b to the challenger. $\mathcal{S}(\mathcal{D}_\omega)$ has the same advantage in the HP-ABE security game, as $\mathcal{D}_\omega$ has in distinguishing between $\mathsf{Game}_{8-(\omega-1)}$ and $\mathsf{Game}_{8-\omega}$.

Proof of indistinguishability between $\mathsf{Game}_{8.5}$ *and* Game_9. We show that if a PPT algorithm $\mathcal{D}$ distinguishes between $\mathsf{Game}_{8.5}$ and Game_9, we can construct an algorithm $\mathcal{S}(\mathcal{D})$ which plays the DDH game in $\mathbb{G}_2$ with non-negligible advantage.

For this we make use of $(N_{\text{users}} N_{\text{records}} + 1)$ hybrid games $\mathsf{Game}_{8.5-0-0}$ to $\mathsf{Game}_{8.5-N_{\text{users}}-N_{\text{records}}}$[4], where in $\mathsf{Game}_{8.5-\phi-\omega}$ the values of P' are computed as in Game_9 for all keys φ and records ψ satisfying $(\varphi, \psi) \leq (\phi, \omega)$ (lexicographical comparison) and computed as in $\mathsf{Game}_{8.5}$ for $(\varphi, \psi) > (\phi, \omega)$. Clearly $\mathsf{Game}_{8.5-0-0}$ is exactly $\mathsf{Game}_{8.5}$, and $\mathsf{Game}_{8.5-N_{\text{users}}-N_{\text{records}}}$ is exactly Game_9.

The existence of $\mathcal{D}$ implies the existence of $\mathcal{D}_{\phi\omega}$ (for some ϕ and some ω) which can distinguish between $\mathsf{Game}_{8.5-\phi-\omega}$ and the lexicographically preceding game with advantage $\frac{1}{N_{\text{users}} N_{\text{records}}}$ that of $\mathcal{D}$. We now show how to construct an algorithm $\mathcal{S}(\mathcal{D}_{\phi\omega})$ which plays the DDH game with advantage at least $\frac{1}{N_{\text{attributes}}} \overset{\text{def}}{=} \frac{1}{\sum_{i=1}^{n} n_i}$ that of $\mathcal{D}_{\phi\omega}$, which is still a non-negligible advantage:

$\mathcal{S}(\mathcal{D}_{\phi\omega})$ begins by choosing a category $j \xleftarrow{\$} \mathbb{N}^*_{n+1}$ and an attribute $j' \xleftarrow{\$} \mathbb{N}_{n_i}$ from that category at random. $\mathcal{S}(\mathcal{D}_{\phi\omega})$ receives a DDH challenge $\left(g_2^{\gamma}, g_2^{\delta}, g_2^{z}\right)$ where z is either $\gamma\delta$ or random. $\mathcal{S}(\mathcal{D}_{\phi\omega})$ generates the public key of the issuer and gives them to $\mathcal{D}_{\phi\omega}$.

$\mathcal{S}(\mathcal{D}_{\phi\omega})$ runs $\mathcal{D}_{\phi\omega}$ for the key issuing phase. $\mathcal{D}_{\phi\omega}$ may choose to be issued up to a polynomial number of keys (under the restriction that $L_0 = 0$). When $\mathcal{S}(\mathcal{D}_{\phi\omega})$ issues the key $\varphi = \phi$: First it checks that $j' = L^{(\phi)}_j$ (else $\mathcal{S}(\mathcal{D}_{\phi\omega})$ aborts and takes a random guess), then it sets $\lambda^{(\phi)}_j = \gamma$ (without being able to compute that value), and finally computes $D^{(\phi)}_{j,1} \leftarrow g_2^{s} (g_2^{\gamma})^{a_{j,j'}}$ and $D^{(\phi)}_{j,2} \leftarrow g_2^{\gamma}$. All other components of the key are computed honestly. For

[4] To simplify notation somewhat, we will ignore the games where no progress is made.

all other keys $\varphi \neq \phi$, $\mathcal{S}(\mathcal{D}_i)$ computes $\lambda_j^{(\varphi)} \xleftarrow{\$} \mathbb{Z}_p^*$, $D_{j,2}^{(\varphi)} \leftarrow g_2^{\lambda_j^{(\varphi)}}$ and $D_{j,1}^{(\varphi)} \leftarrow g_2^s (D_{j,2}^{(\varphi)})^{a_{j,L_j^{(\varphi)}}}$ (so that it knows the discrete logarithm of $D_{j,2}^{(\varphi)}$), and all other key components honestly. ZKPK_5 is now the proof of a false statement.

$\mathcal{S}(\mathcal{D}_{\phi\omega})$ runs $\mathcal{D}_{\phi\omega}$, who chooses a plaintext $K^{(\omega)}$ and ciphertext policy $W^{(\omega)}$. $\mathcal{S}(\mathcal{D}_{\phi\omega})$ aborts if $j' \in W_j^{(\omega)}$.
$\mathcal{S}(\mathcal{D}_{\phi\omega})$ computes the ciphertext:

$$\{\tilde{C}^{(\omega)}, C_0^{(\omega)}, C_{0,1}^{(\omega)}, \{C_{0,0,2}^{(\omega)}, C_{0,\varrho,2}^{(\omega)}\}, \{C_{i,1}^{(\omega)}, \{\underline{Q_{i,t,2}^{(\omega)}}\}_{t=0}^{n_i-1}\}_{i=1}^{n}\}.$$

however instead of computing $Q_{j,j',2}^{(\omega)}$ honestly, it embeds part of the DDH challenge in that value: it sets $Q_{j,j',2}^{(\omega)} = g_1^{\delta}$ (without being able to compute this value since it only knowns g_2^{δ}). All other ciphertext components $Q_{i,t,2}^{(\omega)}$ are computed honestly. Like in the last game however, $\mathcal{S}(\mathcal{D}_{\phi\omega})$ re-randomizes all ciphertext components which are not part of the policy:

$$\begin{aligned} &\forall i \in \mathbb{N}_{n+1}^* : \forall t \in W_i^{(\omega)} : && C_{i,t,2}^{(\omega)} \leftarrow Q_{i,t,2}^{(\omega)}; \\ &\forall i \in \mathbb{N}_{n+1}^* : \forall t \in \mathbb{N}_{n_i}^* \setminus W_i^{(\omega)} : && C_{i,t,2}^{(\omega)} \xleftarrow{\$} \mathbb{G}_1. \end{aligned}$$

and publishes the ciphertext:

$$\{\tilde{C}^{(\omega)}, C_0^{(\omega)}, C_{0,1}^{(\omega)}, \{C_{0,0,2}^{(\omega)}, C_{0,\varrho,2}^{(\omega)}\}, \{C_{i,1}^{(\omega)}, \{\underline{C_{i,t,2}^{(\omega)}}\}_{t=0}^{n_i-1}\}_{i=1}^{n}\}.$$

Note that $\mathcal{S}(\mathcal{D}_{\phi\omega})$ never needs to publish the value $Q_{j,j',2}^{(\omega)}$ it couldn't compute before.

$\mathcal{S}(\mathcal{D}_{\phi\omega})$ generates all records $\psi \neq \omega$ in the same way (including the re-randomization of all ciphertext components $Q_{i,t,2}^{(\psi)}$ not in the policy), except that it does not embed any DDH challenge in them.

The key issuing phase may be repeated.

$\mathcal{D}_{\phi\omega}$ may now perform queries for the record ψ and key φ ($\mathcal{S}(\mathcal{D}_{\phi\omega})$ extracts φ and ψ from ZKPK_4). If the key satisfies the ciphertext policy of the queried record, P' is computed honestly. Else the key doesn't satisfy the policy: if $(\varphi, \psi) < (\phi, \omega)$, P' is computed randomly ; else if $(\varphi, \psi) \geq (\phi, \omega)$, P' is computed as in $\mathsf{Game}_{8.5}$:

$$P' \leftarrow \left(\frac{\hat{C}^{(\psi)} \prod_{i=0}^{n} \mathrm{e}(C_{i,1}^{(\psi)}, D_{i,1}^{(\varphi)})}{K^{(0)}\, \mathrm{e}(C_0^{(\psi)}, D_0^{(\varphi)}) \prod_{i=1}^{n} \mathrm{e}(Q_{i,L_i^{(\varphi)},2}^{(\psi)}, D_{i,2}^{(\varphi)})} \right)^{k_c k_d} . \tag{10}$$

There are however two non-trivial cases of $\mathrm{e}(Q_{j,L_j^{(\varphi)},2}^{(\psi)}, D_{j,2}^{(\varphi)}) \overset{\text{def}}{=} \wp_j^{(\psi,\varphi)}$:

- To compute this pairing for an honest key $\varphi \neq \phi$, but for the record $\psi = \omega$ containing the DDH challenge: since $\lambda_j^{(\varphi)}$ (the discrete logarithm of $D_{j,2}^{(\varphi)}$) is known to $\mathcal{S}(\mathcal{D}_{\phi\omega})$, the pairing can be computed thus: $\wp_j^{(\omega,\varphi)} \leftarrow \mathrm{e}(g_1, g_2^{\delta})^{\lambda_j^{(\varphi)}}$.
- To compute this pairing for the key $\varphi = \phi$ containing the DDH challenge and the record $\psi = \omega$ also containing the DDH challenge, the pairing can be computed thus: $\wp_j^{(\omega,\phi)} = \mathrm{e}(g_1, g_2^{z})$. If z is random, then $\wp_j^{(\omega,\phi)}$ is computed according to $\mathsf{Game}_{8.5-\phi-\omega}$, and if $z = \gamma\delta$, then $\wp_j^{(\omega,\phi)}$ is computed according to the lexicographically preceding game.

Finally $\mathcal{D}_{\phi\omega}$ outputs a guess b of which game it is in. $\mathcal{S}(\mathcal{D}_{\phi\omega})$ returns the same guess b for the DDH challenge. For there to be any difference between the current game and the preceding one, the key ϕ must not satisfy the policy of the record ω. This means at least one attribute in the key is not in the policy of the record; the probability that $\mathcal{S}(\mathcal{D}_{\phi\omega})$ hits such an attribute is therefore at least $N_{\text{attributes}}^{-1}$, and in all other cases it aborts and takes a random guess.

D.7 Proof of lemma 3

If there exists a PPT algorithm $\mathcal{D}$ which distinguishes between Game_9 and Game_{10} with non-negligible advantage, then we can construct a PPT algorithm $\mathcal{S}(\mathcal{D})$ which has rewindable black-box access to $\mathcal{D}$ and which plays the security game of $\mathsf{HP\text{-}ABE}$ with non-negligible advantage.

Again, we consider a series of hybrid games Game_{9-0} to $\mathsf{Game}_{9-N_{\text{records}}}$, where in $\mathsf{Game}_{9-\omega}$ every record $\psi \leq \omega$ is handled as in Game_{10}, and all records $\psi > \omega$ are handled as in Game_9. Clearly, Game_{9-0} is exactly Game_9 and $\mathsf{Game}_{9-N_{\text{records}}}$ is exactly Game_{10}.

The existence of $\mathcal{D}$ implies the existence of $\mathcal{D}_\omega$ (for some ω) that can distinguish between $\mathsf{Game}_{9-(\omega-1)}$ and $\mathsf{Game}_{9-\omega}$ with advantage N_{records}^{-1} that of $\mathcal{D}$.

The challenger computes the issuer public key and sends it to $\mathcal{S}(\mathcal{D}_\omega)$. $\mathcal{S}(\mathcal{D}_\omega)$ relays it to $\mathcal{D}_\omega$.

$\mathcal{S}(\mathcal{D}_\omega)$ runs $\mathcal{D}_\omega$, who may now requests a key with attribute list $\{L_i\}$ where $L_0 = 0$. $\mathcal{S}(\mathcal{D}_\omega)$ relays the request to the challenger and the response back to $\mathcal{D}_\omega$. This can be repeated polynomially many times.

When $\mathcal{D}_\omega$ wants to encrypt a record ψ:

- If $\psi < \omega$, $\mathcal{S}(\mathcal{D}_\omega)$ encrypts a random plaintext $K^{(\psi)\prime} \xleftarrow{\$} \mathbb{G}_{\mathrm{T}}$ under a random policy $W^{(\psi)\prime}$ (but with $W_0^{(\psi)\prime} = \{0\}$). Again, $C_{0,0,2}^{(\psi)}$ is remembered but not published.
- For the challenge record $\psi = \omega$, $\mathcal{S}(\mathcal{D}_\omega)$ sets $K^{(0)} = K^{(\omega)}$ and $W^{(0)} = W^{(\omega)}$, as well as $K^{(1)} \xleftarrow{\$} \mathbb{G}_{\mathrm{T}}$ and selects $W^{(1)}$ randomly (but with $W_0^{(1)} = \{\varrho\}$), and sends these to the $\mathsf{HP\text{-}ABE}$ challenger. The challenger flips a bit $b \xleftarrow{\$} \mathbb{N}_2$ and encrypts $K^{(b)}$ under policy $W^{(b)}$ (we shall denote the resulting ciphertext $C^{(b)}$). $\mathcal{S}(\mathcal{D}_\omega)$ relays the ciphertext to $\mathcal{D}_\omega$.
- If $\psi > \omega$, $\mathcal{S}(\mathcal{D}_\omega)$ encrypts the plaintext $K^{(\psi)}$ under policy $W'^{(\psi)} = \{\{0, \varrho\}, W_1^{(\psi)}, W_2^{(\psi)}, \ldots, W_n^{(\psi)}\}$ (including the “issuer” attribute). $\mathcal{S}(\mathcal{D}_\omega)$ remembers the ciphertext component $C_{0,0,2}^{(\psi)}$ corresponding to the “issuer” attribute but doesn’t publish it. $\mathcal{S}(\mathcal{D}_\omega)$ sends the rest of the ciphertext to $\mathcal{D}_\omega$.

$\mathcal{S}(\mathcal{D}_\omega)$ runs $\mathcal{D}_\omega$, who may now ask for more keys (polynomially many times).

$\mathcal{S}(\mathcal{D}_\omega)$ runs $\mathcal{D}_\omega$, who may perform a query with a key φ on record ψ ($\mathcal{S}(\mathcal{D}_\omega)$ extracts φ and ψ from ZKPK_6). $\mathcal{S}(\mathcal{D}_\omega)$ checks if the key φ satisfies the ciphertext policy $W^{(\psi)}$. If the key doesn’t satisfy the policy, then $\mathcal{S}(\mathcal{D}_\omega)$ computes P' randomly. P' is thus distributed exactly as in both games. If it does then:

- For records $\psi < \omega$, P' is computed as follows:

$$P' \leftarrow \left(\frac{\hat{C}^{(\psi)} \prod_{i=0}^{n} \mathrm{e}(C_{i,1}^{(\psi)}, D_{i,1}^{(\varphi)})}{K^{(\psi)}\, \mathrm{e}(C_0^{(\psi)}, D_0^{(\varphi)}) \prod_{i=1}^{n} \mathrm{e}(C_{i,L_i^{(\varphi)},2}^{(\psi)}, D_{i,2}^{(\varphi)})} \right)^{k_c k_d} . \tag{11}$$

- For records $\psi = \omega$, P' is computed as follows:

$$P' \leftarrow \left(\frac{\hat{C}^{(b)} \prod_{i=0}^{n} \mathrm{e}(C_{i,1}^{(b)}, D_{i,1}^{(\varphi)})}{K^{(0)}\, \mathrm{e}(C_0^{(b)}, D_0^{(\varphi)}) \prod_{i=1}^{n} \mathrm{e}(C_{i,L_i^{(\varphi)},2}^{(b)}, D_{i,2}^{(\varphi)})} \right)^{k_c k_d} . \tag{12}$$

If $b = 0$ then $P' = \mathrm{e}(C_{0,\varrho,2}^{(\omega)}, D_{0,2}^{(\varphi)})^{k_\varrho^{-1} k_c k_d}$ and thus distributed exactly as in $\mathsf{Game}_{9-(\omega-1)}$. If $b = 1$ then P' is distributed exactly as in $\mathsf{Game}_{9-\omega}$.

- For records $\psi > \omega$, P' is computed as:

$$P' \leftarrow \mathrm{e}(C_{0,0,2}^{(\psi)}, D_{0,2}^{(\varphi)})^{k_c k_d}.$$

$\mathcal{S}(\mathcal{D}_\omega)$ sends the result P' back to $\mathcal{D}_\omega$.

The query phase may be repeated polynomially many times.

$\mathcal{S}(\mathcal{D}_\omega)$ runs $\mathcal{D}_\omega$, who now finally outputs a guess of b. $\mathcal{S}(\mathcal{D}_\omega)$ sends b to the challenger. $\mathcal{S}(\mathcal{D}_\omega)$ has the same advantage in the $\mathsf{HP\text{-}ABE}$ security game, as $\mathcal{D}_\omega$ has in distinguishing between $\mathsf{Game}_{9-(\omega-1)}$ and $\mathsf{Game}_{9-\omega}$.

D.8 Limitations

Note that the set of corrupted parties is fixed before the start of the protocol. Our scheme cannot be proven secure if we allow dynamic corruptions (even with erasures): the underlying $\mathsf{HP\text{-}ABE}$ scheme is not receiver non-committing, and as $\mathcal{S}$ may be required to publish a ciphertext containing a bogus plaintext and policy on behalf of an honest database, $\mathcal{S}$ will be caught when that database becomes corrupted.

Supporting unlimited-length plaintexts $M \in \{0,1\}^*$ in the ideal world causes a problem: $\mathcal{S}$ may be required to publish an authenticated ciphertext containing a bogus plaintext on behalf of an honest database, $\mathcal{S}$ will be caught when a user decrypts the bogus authenticated ciphertext. (We could solve this issue in Canetti's Universal Composability model in a similar way that he handles UC-encryption: by publishing the authenticated ciphertext in the ideal functionality. However this approach suffers from high degree of complexity and is tricky to define correctly.)

We further note that unless the simulator $\mathcal{S}$ can extract the witnesses from zero-knowledge proofs and perform proofs of incorrect statements without rewinding $\mathcal{A}$ and $\mathcal{E}$, our results will not hold in the Universal Composability setting. In this respect, our security model is similar to the one used in [CNS07,CDN09,CDNZ11].

E Comparison with CDNZ (Without Revocation)

Table 1. Comparison of the communication costs borne by the user and database when receiving records and during the transfer step. The size estimates are theoretical and only take into account the size of the group elements. We let $\kappa = \log_2(p)$. We use ℓ to denote the number of categories in CDNZ [CDNZ11]; we use n to denote the number of categories, and V the number of attributes in this paper.

Item	Paper	Elements in $\mathbb{Z}_p$	Elem. in $\mathbb{G}_1$	Elem. in $\mathbb{G}_2$	Elem. in $\mathbb{G}_T$	Bits transfered[a]
Receive record (DB→user)[b]	This work	1	$V + 2n + 11$	$2n + 4$	1	$(V + 8n + 30)\kappa$
Receive record (DB→user)	CDNZ	—	$4\ell + 2$	$4\ell + 5$	1	$(16\ell + 23)\kappa$
Query (user↔DB)	This work	19	18	14	21	205κ
Query (user↔DB)	CDNZ	$6\ell + 55$	$4\ell + 40$	$4\ell + 7$	17	$(22\ell + 218)\kappa$

[a] When ignoring overhead. Assuming D-type curves as used by the PBC library [Lyn07]: κ, κ, 3κ, 6κ bits per element in $\mathbb{Z}_p$, $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ respectively. For the curve `d159`: $\kappa = 160$.

[b] Note that the storage cost of records is the same same as the communication cost.

Table 2. Comparison of the computational costs borne by the user and database when receiving records and during the transfer step. The run-time estimates are theoretical when considering only exponentiations and pairings. We use ℓ to denote the number of categories in CDNZ [CDNZ11]; we use n to denote the number of categories, and V the number of attributes in this paper.

Item	Paper	Expon. $\mathbb{G}_1$	Expon. $\mathbb{G}_2$	Expon. $\mathbb{G}_T$	Pairings	Runtime[a]
Generate record (DB)	This work	$V + 4n + 16$	$4n + 10$	1	—	$V + 40n + 108$
Generate record (DB)	CDNZ	$4\ell + 7$	$8\ell + 9$	—	1	$76\ell + 97$
Receive/check record (user)	This work	—	4	—	$8n + 26$	$72n + 270$
Receive/check record (user)	CDNZ	—	—	—	$16\ell + 10$	$144\ell + 90$
Query (user)	This work	27	22	38	$2n + 43$	$18n + 688$
Query (user)	CDNZ	$16\ell + 99$	$8\ell + 35$	$3\ell + 36$	1	$94\ell + 495$
Query (DB)	This work	17	9	45	41	557
Query (DB)	CDNZ	$12\ell + 104$	—	$4\ell + 51$	$18\ell + 20$	$182\ell + 386$

[a] We take the time to do one exponentiation in $\mathbb{G}_1$ as the base unit. Tests on different hardware showed that for groups of 159-bit order (the `d159` curve of the PBC Library [Lyn07]), exponentiations in $\mathbb{G}_T$ are twice as slow as in $\mathbb{G}_1$, and that pairings and exponentiations in $\mathbb{G}_2$ are nine times slower. We ignore additions, multiplications, inversions and overhead. On a Core i7 CPU clocked at 1600 MHz, one exponentiation in $\mathbb{G}_1$ on the `d159` curve takes approximately 1.49 milliseconds.

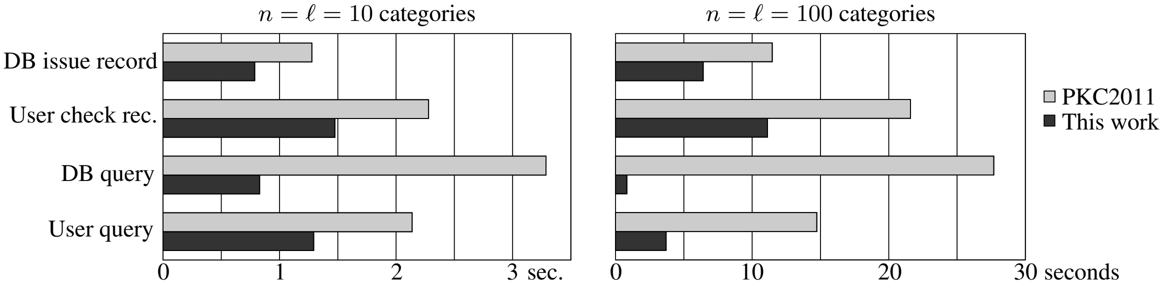

Fig. 12. Theoretical runtime comparison, when using CDNZ-policies with our scheme ($V = 2n$; $n = \ell$). The runtimes are computed according to table 2, assuming we use the `d159` curve in the PBC Library [Lyn07], and perform the computations on a Core i7 CPU clocked at 1600 MHz.