Des. Codes Cryptogr. (2016) 81:179–223
DOI 10.1007/s10623-015-0134-y

Blind attribute-based encryption and oblivious transfer with fine-grained access control

Alfredo Rial[1]

Received: 1 August 2012 / Revised: 4 September 2015 / Accepted: 7 September 2015 /
Published online: 1 October 2015

Abstract We propose two constructions of oblivious transfer with access control (OTAC), i.e., oblivious transfer schemes in which a receiver can obtain a message only if her attributes, which are certified by a credential issuer, satisfy the access control policy of that message. The receiver remains anonymous towards the sender and the receiver's attributes are not disclosed to the sender. Our constructions are based on any ciphertext policy attribute based encryption (CPABE) scheme that fulfills the committing and key separation properties, which we define. We also provide a committing CPABE with key separation scheme that supports any policy described by a monotone access structure, which, in comparison to previous work, allows our OTAC construction to support efficiently a wider variety of access control policies. In our constructions, a receiver obtains from the sender a CPABE secret key for her attributes by using a blind key extraction with access control protocol. We provide a blind key extraction with access control protocol for any committing CPABE with key separation scheme. Previous work only provided ad-hoc constructions of blind key extraction protocols. Our generic protocol works in a hybrid model that employs novel ideal functionalities for oblivious transfer and for anonymous attribute authentication. We propose constructions that realize those novel ideal functionalities and analyze the overall efficiency of our OTAC constructions.

Keywords Oblivious transfer · Attribute-based encryption · Access control · Blind key extraction · Cryptographic protocols · Privacy

Mathematics Subject Classification 94A60 · 68P25

Communicated by C. Padro.

✉ Alfredo Rial
lia@zurich.ibm.com

[1] IBM Research Zurich, Rueschlikon, Switzerland

1 Introduction

Oblivious transfer [25] (OT) is a cryptographic primitive that allows the construction of privacy-preserving databases, i.e., databases that allow users to query and retrieve information anonymously and without the data holder learning which information is accessed. Thanks to these properties, oblivious transfer has been proposed to solve privacy-related problems in a wide variety of applications that involve sensitive information, such as medical databases, patent searches or location-based services [21,22].

However, current regulations (e.g., HIPAA) require data holders to enact strict accounting procedures in order to ensure that data is only accessed by authorized parties. Therefore, it is desirable to provide databases based on OT with privacy-preserving access control mechanisms that allow the data holder to enforce access control policies. The current trend towards distributing and outsourcing sensitive information (e.g. Microsoft HealthVault) makes this requirement even more compelling.

Privacy-preserving access control mechanisms for OT must define, for each record of data, an access control policy that determines the attributes, rights or roles that a user must possess in order to be granted access. However, this should be done without harming the privacy properties of OT, i.e., users must remain anonymous towards the data holder and must still be able to hide from him both their attributes and the records they access. Additionally, the access control mechanism should be flexible enough to allow the data holder to apply a wide variety of access control policies.

1.1 Previous work

K-out-of-N [9] OT is a two-party protocol between a sender and a receiver. The sender receives as input N messages $(m_1, \ldots, m_N)$, while the receiver gets K selection values $(\sigma_1, \ldots, \sigma_K)$. As output, the receiver gets the messages $(m_{\sigma_1}, \ldots, m_{\sigma_K})$. Sender privacy requires that the receiver gets no information on the other messages, while receiver privacy requires that the sender does not learn any information on $(\sigma_1, \ldots, \sigma_K)$. In adaptive OT, the receiver chooses σ_i ($i \in [1, K]$) after outputting message $m_{\sigma_{i-1}}$, while, in non-adaptive OT, the selection values $(\sigma_1, \ldots, \sigma_K)$ are chosen before outputting any message.

Oblivious transfer with access control (OTAC) [6,11] allows the sender to control access to the messages. The sender receives as input $(m_1, \mathbb{P}_1, \ldots, m_N, \mathbb{P}_N)$, where $(\mathbb{P}_1, \ldots, \mathbb{P}_N)$ are access control policies for each of the messages. Each receiver possesses a set of attributes $\mathbb{A}$, which is certified by a credential issuer, and is able to obtain the message m_i only if $\mathbb{A}$ satisfies $\mathbb{P}_i$. Receiver privacy requires that the sender does not get any information on $\mathbb{A}$.

Coull et al. [11] propose a scheme in which access control policies are described by a state graph, such that each state allows access to a subset of the records. In the initialization phase, the database records are encrypted on input an index from 1 to N. Additionally, each user obtains a credential that binds her to a state in the graph. To access a record, the user proves possession of her credential and proves that the index of the record is included in the subset of indices defined by her state. Each access to the database determines a transition from one state to another, which depends on the record accessed. The user obtains a new credential that binds her to her new state. Access control is enforced by limiting the possible transitions between states.

This scheme has the nice properties that it can be applied to different OT schemes and that it permits changing the policies without needing to rerun the initialization phase of the OT scheme. However, as noted by Camenisch et al. [6], the scheme is inefficient for two reasons.

First, users must obtain a new credential each time they access the database. Second, a large class of access control policies cannot be efficiently expressed via state graphs.

Camenisch et al. [6] modify the OT scheme in [9] to provide access control. First, each user obtains a credential that certifies that she possesses some attributes. In the initialization phase, the sender encrypts each record of data on input a set of attributes. To access a record, the user must prove that she possesses a credential that contains all the record's attributes. The access control policy class that can be expressed is therefore limited to conjunction of attributes. To allow expressing disjunction, the database holder can replicate the record, increasing thus the database size. For example, let (a_1, a_2, a_3, a_4) be attributes and $(a_1 \wedge a_2) \vee (a_3 \wedge a_4)$ be the policy for record m. The sender can input to the database two records $m_1 = m_2 = m$, where the policy for m_1 is $(a_1 \wedge a_2)$ and, for m_2, $(a_3 \wedge a_4)$. A more recent work [1], also limited to conjunctive policies, achieves universal composability.

After the work by Camenisch et al. [6], other schemes aim at achieving better efficiency or at supporting more expressive access control policies. To achieve better efficiency, the access control policies in the protocol in [31] are limited to proving possession of one attribute. In the protocol in [20], the user authenticates herself using an anonymous credential, but the messages are not associated to access control policies.

To support more expressive access control policies, the protocol in [30] employs the fuzzy identity based encryption scheme by Sahai and Waters [27] in order to support threshold policies. In threshold policies, each record is associated with a set of attributes and a user is granted access if the number of attributes in the policy that she possesses is over a threshold.

Other protocols go further and employ ciphertext policy attribute based encryption (CPABE) [17–19,26,29,32]. Generally speaking, these protocols work as follows. In the initialization phase, each record of data is associated with an access control policy. The sender encrypts the records of data under their respective policies via the CPABE scheme. To access a record, the user must hold a secret key whose attributes fulfill the record's access control policy.

In some schemes [18,19,32], users must reveal their attributes in order to obtain a secret key of the CPABE scheme. We note that in [6] the credential issuer does not necessarily learn the users' attributes. Namely, the credential issuer can certify the attributes by letting the user prove in zero-knowledge statements about them.

To solve this problem, an earlier version of this work [26] and other schemes [17,29] propose the use of a blind key extraction protocol for the CPABE scheme. The protocol in [17] employs blind CPABE but the protocol involves only a sender and a receiver, who obtains the secret key for the CPABE scheme engaging with a sender in a blind key extraction protocol. However, the attributes of the user are not certified, so it is unclear how access control is actually enforced.

The protocol in [29] employs blind CPABE along with a credential scheme. The concrete blind CPABE proposed supports policies that are described by conjunctions and disjunctions of attributes, but threshold policies are not efficiently supported. Additionally, the parameters of the CPABE scheme grow with the number of possible attributes and the CPABE is not proven to be committing, a property that ensures that a corrupt sender cannot compute malformed ciphertexts that decrypt to different messages. Finally, although a concrete blind ABE is proposed, no generic way of constructing blind CPABE with access control protocols for any CPABE scheme is proposed.

Prior to [17,29], an earlier version of this work [26] proposed a construction of OT with access control from any CPABE scheme, OT and anonymous credentials scheme. The concrete CPABE scheme used supports efficiently any policy that can be expressed by a monotonic access structure, which includes conjunctive, disjunctive and threshold policies,

and, although the ciphertext size grows with the size of the access structure, the size of the public parameters is independent of the number of attributes. In [26], a generic way of building blind CPABE with access control from CPABE is proposed, which employs OT and anonymous credentials. However, although security for OT and anonymous credentials is defined via an ideal functionality, the proposed protocol does not employ these functionalities as building blocks in a hybrid protocol, which hinders proving security of the construction.

In a related line of work, Camenisch et al. [7] propose a scheme where access control policies are hidden from the user. However, the access control policy class is limited to conjunction of attributes. In [5], Camenisch et al. improve the work in [7]. They propose an OT with hidden access control policies scheme based on the hidden ciphertext-policy attribute based-encryption by Nishide et al. [23]. The access control policy class supported is described as follows. Let n be the number of attribute types. The set S_i ($i \in [1, n]$) contains the possible values that an attribute s_i of type i can have. A receiver possesses a set of attributes $(s_1, \ldots, s_n)$ such that $s_i \in S_i$ ($i \in [1, n]$). An access control policy is given by $(W_1, \ldots, W_n)$, where $W_i \subseteq S_i$ is a subset of S_i. The policy is satisfied if, for $i = 1$ to N, $s_i \in W_i$. As can be seen, to express disjunction of attributes of different categories, the sender must also replicate messages. Moreover, the credential issuer necessarily learns the attributes to be able to compute a secret key of the CPABE scheme.

1.2 Our contribution

We propose a non-restricted and a restricted OTAC scheme. In a non-restricted OTAC scheme, the receiver can obtain in one transfer phase all the messages whose access control policy is fulfilled by the receiver's attributes. In a restricted OTAC scheme, the receiver can only obtain one message per transfer phase.

Our schemes employ CPABE, a non-adaptive OT functionality, and a simple anonymous credentials functionality referred to as anonymous attribute authentication. The restricted OTAC construction also employs an adaptive OT functionality.

The sender employs committing CPABE to encrypt the messages under their respective access control policies. We define the committing property, which ensures that malformed ciphertexts cannot yield different messages.

To access the messages, a receiver gets a CPABE secret key for her attributes from the sender by using a blind key extraction with access control protocol. We provide a generic construction of a blind key extraction with access control protocol that works for any committing CPABE scheme that fulfills an additional property we call key separation. Basically, key separation guarantees that a secret key for attributes $(a_1, \ldots, a_L)$ consists of $(sk_{a_1}, \ldots, sk_{a_L}, sk'_{\mathbb{A}})$, where each sk_{a_l} is computed on input the attribute a_l only, and $sk'_{\mathbb{A}}$ is computed without knowledge of any attribute.

Our generic construction of blind key extraction with access control works in a hybrid model that employs two novel ideal functionalities for non-adaptive OT and for anonymous attribute authentication. We consider a universe of attributes $[1, L_{uni}]$, where L_{uni} is the size of the universe. The sender inputs a secret key $(sk_{a_1}, \ldots, sk_{a_{L_{uni}}})$ to the non-adaptive OT functionality, where each sk_{a_l} acts as a message to be transferred. The issuer issues some attributes $(a_1, \ldots, a_L)$ to a receiver through the anonymous attribute authentication functionality. In order to obtain a secret key for the attributes issued, the receiver must prove to the sender that these attributes were issued by the issuer through the anonymous attribute authentication functionality. Additionally, the receiver must prove that the selection values $(\sigma_1, \ldots, \sigma_K)$ sent as input to the non-adaptive OT functionality equal the attributes sent as input to the anonymous attribute authentication functionality.

However, existing ideal functionalities for non-adaptive OT do not allow for that, i.e., in a hybrid protocol where the non-adaptive OT functionality is used along with other functionalities, existing non-adaptive OT functionalities do not provide a means that allows the sender to check whether the selection values input to the non-adaptive OT functionality equal those input to other functionalities. To solve this problem, we propose a novel non-adaptive OT functionality and a novel anonymous attribute authentication functionality where the receiver sends committed inputs. The sender receives those commitments from the functionalities and checks their equality. Under the binding property of the commitment scheme, it holds that the input to both functionalities is equal.[1]

We propose a concrete committing CPABE with key separation scheme based on the CPABE scheme in [3]. We also show constructions that realize our novel non-adaptive OT and anonymous attribute authentication functionalities. We define security of non-restricted and restricted OTAC protocols in the universal composability framework and we prove that our protocols for non-restricted and restricted OTAC realize their respective functionalities. The constructions we propose do not achieve universal composability because, for the sake of efficiency, we employ the Fiat-Shamir transform [13] to instantiate non-interactive zero-knowledge proofs.

1.3 Outline of the paper

Section 2 is devoted to technical preliminaries. In Sect. 3, we describe the concept of committing CPABE with key separation. Section 4 describes briefly the universal composability model and describes the ideal functionalities that our OTAC protocols employ as building block. In particular, we propose novel functionalities for anonymous attribute authentication and for non-adaptive OT. In Sect. 5, we define ideal functionalities for non-restricted and restricted OTAC and describe our non-restricted and restricted OTAC construction. In Sect. 6, we propose concrete constructions for the building blocks of our OTAC protocols. In particular, we describe a construction of committing CPABE with key separation, a construction of anonymous attribute authentication, and a construction of non-adaptive OT. We conclude in Sect. 7.

2 Preliminaries

2.1 Bilinear maps

Let $\mathbb{G}$, $\tilde{\mathbb{G}}$ and $\mathbb{G}_t$ be groups of prime order p. A map $e : \mathbb{G} \times \tilde{\mathbb{G}} \to \mathbb{G}_t$ must satisfy bilinearity, i.e., $e(g^x, \tilde{g}^y) = e(g, \tilde{g})^{xy}$; non-degeneracy, i.e., for all generators $g \in \mathbb{G}$ and $\tilde{g} \in \tilde{\mathbb{G}}$, $e(g, \tilde{g})$ generates $\mathbb{G}_t$; and efficiency, i.e., there exists an efficient algorithm $\mathcal{G}(1^k)$ that outputs the pairing group setup $(p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_t, e, g, \tilde{g})$ and an efficient algorithm to compute $e(a, b)$ for any $a \in \mathbb{G}, b \in \tilde{\mathbb{G}}$. If $\mathbb{G} = \tilde{\mathbb{G}}$ the map is symmetric and otherwise asymmetric.

2.2 Access structures and access trees

Let $\{s_1, s_2, \ldots, s_n\}$ be a set of attributes. A collection $\mathbb{P} \subseteq 2^{\{s_1, s_2, \ldots, s_n\}}$ is monotone if, $\forall$ B and C, if $B \in \mathbb{P}$ and $B \subseteq C$ then $C \in \mathbb{P}$. An access structure (respectively, monotone

[1] The idea of using a commitment scheme for this purpose is taken from "UC Commitments, Revocation, and Attribute Tokens for Privacy-Preserving Protocol Design" by Jan Camenisch, Maria Dubovitskaya and Alfredo Rial.

access structure) is a collection (respectively, monotone collection) $\mathbb{P}$ of non-empty subsets of $\{s_1, s_2, \ldots, s_n\}$, i.e., $\mathbb{P} \subseteq 2^{\{s_1, s_2, \ldots, s_n\}} \backslash \emptyset$. The sets in $\mathbb{P}$ are called the authorized sets, and the sets not in $\mathbb{P}$ are called the unauthorized sets.

Let $\mathcal{T}$ be a tree representing an access structure. Each non-leaf node x represents a threshold gate described by an amount n_x of children and a threshold value k_x such that $0 < k_x \leq n_x$. (When $k_x = 1$, x represents an OR gate and, when $k_x = n_x$, x represents an AND gate.) Each leaf node x is described by an attribute.

We denote by $\mathsf{parent}(x)$ the parent of node x, and by $\mathsf{att}(x)$ the attribute associated with the leaf node x. The children of every node are ordered from 1 to n_x. $\mathsf{index}(x)$ returns such a number associated with the node x.

Let $\mathcal{T}$ be an access tree with root r, and $\mathcal{T}_x$ the subtree of $\mathcal{T}$ rooted at x (hence $\mathcal{T}$ equals $\mathcal{T}_r$). $\mathcal{T}_x(\mathbb{A})$ is a function that returns 1 if the set of attributes $\mathbb{A}$ satisfies $\mathcal{T}_x$. $\mathcal{T}_x(\mathbb{A})$ is computed recursively as follows. If x is a non-leaf node, evaluate $\mathcal{T}_{x'}$ for all children x' of node x. $\mathcal{T}_x(\mathbb{A})$ returns 1 when at least k_x children return 1. If x is a leaf node, $\mathcal{T}_x(\mathbb{A})$ returns 1 when $\mathsf{att}(x) \in \mathbb{A}$.

2.3 Zero-knowledge proofs of knowledge

Informally speaking, a zero-knowledge proof of knowledge is a two-party protocol between a prover and a verifier with two properties. First, it should be a proof of knowledge, i.e., there should exist a knowledge extractor that extracts the secret input from a successful prover with all but negligible probability. Second, it should be zero-knowledge, i.e., for all possible verifiers there exists a simulator that, without knowledge of the secret input, yields a distribution that cannot be distinguished from the interaction with a real prover.

To express a zero-knowledge proof of knowledge, we follow the notation introduced by Camenisch and Stadler [4]. For example, $\mathsf{PK}\{(x) : y = f(x)\}$ denotes a "*zero-knowledge proof of knowledge of the secret input x such that $y = f(x)$*". Letters in the parenthesis, in this example x, denote the secret input, while y and the function f are also known to the verifier.

Let $\mathcal{L}$ be a language in NP. We can associate to any NP-language $\mathcal{L}$ a polynomial time recognizable relation $\mathcal{R}_{\mathcal{L}}$ defining $\mathcal{L}$ as $\mathcal{L} = \{x : \exists w \text{ s.t. } (x, w) \in \mathcal{R}_{\mathcal{L}}\}$, where $|w| \leq \text{poly}(|x|)$. The string w is called a witness for membership of $x \in \mathcal{L}$.

A protocol $\Sigma = (\mathcal{P}, \mathcal{V})$ for an NP-language $\mathcal{L}$ is an interactive proof system. The prover $\mathcal{P}$ and the verifier $\mathcal{V}$ know an instance x of the language $\mathcal{L}$. The prover $\mathcal{P}$ also knows a witness w for membership of $x \in \mathcal{L}$. Σ-protocols have a 3-move shape where the first message α, called commitment, is sent by the prover. The second message β, called challenge, is chosen randomly and sent by the verifier. The last message γ, called response, is sent by the prover. A Σ-protocol fulfills the properties of completeness, honest-verifier zero-knowledge, and special soundness defined in Faust et al. [12].

In our constructions, we employ zero-knowledge proofs of knowledge based on the Fiat-Shamir transform [13]. The Fiat-Shamir transform removes the interaction between the prover $\mathcal{P}$ and the verifier $\mathcal{V}$ of a Σ protocol by replacing the challenge with a hash value $H(\alpha, x)$ computed by the prover, where H is modeled as a random oracle. A proof π consists of $(\alpha, H(\alpha, x), \gamma)$. The Fiat-Shamir proof system is denoted by $(\mathcal{P}^H, \mathcal{V}^H)$ and fulfills the properties of zero-knowledge and weak simulation extractability defined in Faust et al. [12].

Definition 1 (*Zero-Knowledge*) Define the zero knowledge simulator $\mathcal{S}$ as follows. $\mathcal{S}$ is a stateful algorithm that can operate in two modes: $(h_i, st) \leftarrow \mathcal{S}(1, st, q_i)$ answers random

oracle queries q_i, while $(\pi, st) \leftarrow \mathcal{S}(2, st, x)$ outputs a simulated proof π for an instance x. $\mathcal{S}(1, \cdots)$ and $\mathcal{S}(2, \cdots)$ share the state st that is updated after each operation.

Let $\mathcal{L}$ be a language in NP. Denote with $(\mathcal{S}_1, \mathcal{S}_2)$ the oracles such that $\mathcal{S}_1(q_i)$ returns the first output of $(h_i, st) \leftarrow \mathcal{S}(1, st, q_i)$ and $\mathcal{S}_2(x, w)$ returns the first output of $(\pi, st) \leftarrow \mathcal{S}(2, st, x)$ if $(x, w) \in \mathcal{R}_{\mathcal{L}}$. A protocol $(\mathcal{P}^H, \mathcal{V}^H)$ is a non-interactive zero-knowledge proof for the language $\mathcal{L}$ in the random oracle model if there exists a ppt simulator $\mathcal{S}$ such that for all ppt distinguishers $\mathcal{D}$ we have

$$\Pr\left[\mathcal{D}^{H(\cdot),\mathcal{P}^H(\cdot,\cdot)}\left(1^k\right) = 1\right] \approx \Pr\left[\mathcal{D}^{\mathcal{S}_1(\cdot),\mathcal{S}_2(\cdot,\cdot)}\left(1^k\right) = 1\right],$$

where both $\mathcal{P}$ and $\mathcal{S}_2$ oracles output $\perp$ if $(x, w) \notin \mathcal{R}_{\mathcal{L}}$.

Definition 2 (*Weak Simulation Extractability*) Let $\mathcal{L}$ be a language in NP. Consider a non-interactive zero-knowledge proof system $(\mathcal{P}^H, \mathcal{V}^H)$ for $\mathcal{L}$ with zero-knowledge simulator $\mathcal{S}$. Let $(\mathcal{S}_1, \mathcal{S}_2')$ be oracles returning the first output of $(h_i, st) \leftarrow \mathcal{S}(1, st, q_i)$ and $(\pi, st) \leftarrow \mathcal{S}(2, st, x)$ respectively. $(\mathcal{P}^H, \mathcal{V}^H)$ is weakly simulation extractable with extraction error ν and with respect to $\mathcal{S}$ in the random oracle model, if for all ppt adversaries $\mathcal{A}$ there exists an efficient algorithm $\mathcal{E}_{\mathcal{A}}$ with access to the answers $(\mathcal{T}_H, \mathcal{T})$ of $(\mathcal{S}_1, \mathcal{S}_2')$ respectively such that the following holds. Let

$$\begin{aligned}
\text{acc} &= \Pr\left[\left(x^*, \pi^*\right) \leftarrow \mathcal{A}^{\mathcal{S}_1(\cdot),\mathcal{S}_2'(\cdot)}\left(1^k; \rho\right) : \left(x^*, \pi^*\right) \notin \mathcal{T}; \mathcal{V}^{\mathcal{S}_1}\left(x^*, \pi^*\right) = 1\right] \\
\text{ext} &= \Pr\left[\left(x^*, \pi^*\right) \leftarrow \mathcal{A}^{\mathcal{S}_1(\cdot),\mathcal{S}_2'(\cdot)}\left(1^k; \rho\right);\right. \\
&\qquad \left. w^* \leftarrow \mathcal{E}_{\mathcal{A}}\left(x^*, \pi^*; \rho, \mathcal{T}_H, \mathcal{T}\right) : \left(x^*, \pi^*\right) \notin \mathcal{T}; \left(x^*, w^*\right) \in \mathcal{R}_{\mathcal{L}}\right],
\end{aligned}$$

where the probability space in both cases is over the random choices of $\mathcal{S}$ and the adversary's random tape ρ. Then, there exists a constant $d > 0$ and a polynomial p such that whenever $\text{acc} \geq \nu$, we have $\text{ext} \geq (1/p)(\text{acc} - \nu)^d$.

2.4 Commitment schemes

A commitment scheme consists of algorithms CSetup, Com and VfCom. The algorithm $\mathsf{CSetup}(1^k)$ generates the parameters of the commitment scheme par_c, which include a description of the message space $\mathcal{M}$. $\mathsf{Com}(par_c, x)$ outputs a commitment com to x and auxiliary information $open$. A commitment is opened by revealing $(x, open)$ and checking whether $\mathsf{VfCom}(par_c, com, x, open)$ outputs 1 or 0.

A commitment scheme should fulfill the correctness, hiding and binding properties. Correctness requires that VfCom accepts all commitments created by algorithm Com, i.e., for all $x \in \mathcal{M}$

$$\Pr\left[\begin{array}{c} par_c \leftarrow \mathsf{CSetup}(1^k);\ (com, open) \leftarrow \mathsf{Com}(par_c, x) : \\ 1 = \mathsf{VfCom}(par_c, com, x, open) \end{array}\right] = 1.$$

The hiding property ensures that a commitment com to x does not reveal any information about x, whereas the binding property ensures that com cannot be opened to another value x'.

Definition 3 (*Hiding Property*) For any PPT adversary $\mathcal{A}$, the hiding property is defined as follows:

$$\Pr\left[\begin{array}{l} par_c \leftarrow \mathsf{CSetup}(1^k); \\ (x_0, st) \leftarrow \mathcal{A}(par_c); \\ x_1 \leftarrow \mathcal{M}; \\ b \leftarrow \{0,1\};\ (com, open) \leftarrow \mathsf{Com}(par_c, x_b); \\ b' \leftarrow \mathcal{A}(st, com) : \\ x_0 \in \mathcal{M} \ \wedge\ x_1 \in \mathcal{M} \ \wedge\ b = b' \end{array}\right] \leq \frac{1}{2} + \epsilon(k)\ .$$

Definition 4 (*Binding Property*) For any PPT adversary $\mathcal{A}$, the binding property is defined as follows:

$$\Pr\left[\begin{array}{l} par_c \leftarrow \mathsf{CSetup}(1^k);\ (com, x, open, x', open') \leftarrow \mathcal{A}(par_c) : \\ x \in \mathcal{M} \ \wedge\ x' \in \mathcal{M} \ \wedge x \neq x' \wedge\ 1 = \mathsf{VfCom}(par_c, com, x, open) \\ \wedge\ 1 = \mathsf{VfCom}(par_c, com, x', open') \end{array}\right] \leq \epsilon(k)\ .$$

We employ the commitment scheme by Pedersen [24] to commit to elements $x \in \mathbb{Z}_p$, where p is a prime. This commitment scheme is perfectly hiding and computationally binding under the discrete logarithm assumption. The Pedersen commitment scheme consists of the following algorithms.

$\mathsf{CSetup}(1^k)$	On input the security parameter 1^k, pick random generators g, h of a group $\mathbb{G}_p$ of prime order p. Output $par_c = (g, h, \mathcal{M})$, where $\mathcal{M} = \mathbb{Z}_p$.
$\mathsf{Com}(par_c, x)$	Check that $x \in \mathcal{M}$. Pick random $open \leftarrow \mathbb{Z}_p$. Compute $com = h^{open} g^x$ and output com.
$\mathsf{VfCom}(par_c, com, x', open')$	Compute $com' = h^{open'} g^{x'}$. If $com = com'$ then output 1 else 0.

2.5 Signature schemes

A signature scheme consists of the algorithms KeyGen, Sign, and VfSig. Algorithm $\mathsf{KeyGen}(1^k)$ outputs a secret key sk and a public key pk, which include a description of the message space $\mathcal{M}$. $\mathsf{Sign}(sk, m)$ outputs a signature s on message $m \in \mathcal{M}$. $\mathsf{VfSig}(pk, s, m)$ outputs 1 if s is a valid signature on m and 0 otherwise. This definition can be extended to blocks of messages $\bar{m} = (m_1, \ldots, m_n)$. In this case, $\mathsf{KeyGen}(1^k, n)$ receives the maximum number of messages as input.

A signature scheme must fulfill the following correctness and existential unforgeability properties [14].

Definition 5 (*Correctness*) Correctness ensures that the algorithm VfSig accepts the signatures created by the algorithm Sign on input a secret key computed by algorithm KeyGen. More formally, correctness is defined as follows.

$$\Pr\left[\begin{array}{l} (sk, pk) \xleftarrow{\$} \mathsf{KeyGen}(1^k);\ m \xleftarrow{\$} \mathcal{M}; \\ s \xleftarrow{\$} \mathsf{Sign}(sk, m) :\ 1 = \mathsf{VfSig}(pk, s, m) \end{array}\right] = 1.$$

Definition 6 (*Existential Unforgeability*) The property of existential unforgeability ensures that it is not feasible to output a signature on a message without knowledge of the secret key or of another signature on that message. Let $\mathcal{O}_s$ be an oracle that, on input sk and a message

$m \in \mathcal{M}$, outputs $\mathsf{Sign}(sk, m)$, and let S_s be a set that contains the messages sent to $\mathcal{O}_s$. More formally, for any ppt adversary $\mathcal{A}$, existential unforgeability is defined as follows.

$$\Pr\left[\begin{array}{l}(sk, pk) \xleftarrow{\$} \mathsf{KeyGen}(1^k);\ (m, s) \xleftarrow{\$} \mathcal{A}(pk)^{\mathcal{O}_s(sk,\cdot)}: \\ 1 = \mathsf{VfSig}(pk, s, m)\ \wedge\ m \in \mathcal{M}\ \wedge\ m \notin S_s\end{array}\right] \leq \epsilon(k).$$

We employ the signature scheme proposed by Au et al. [2]. This scheme is secure under the strong Diffie-Hellman assumption. Let n be the maximum number of messages that can be signed. The signature scheme in [2] consists of the following algorithms.

$\mathsf{KeyGen}(1^\kappa, n)$	Run $\mathcal{G}(1^\kappa)$ to obtain a pairing group setup $(p, \mathbb{G}, \mathbb{G}_t, e, g)$. Pick random bases $g_1, h_1, \ldots, h_{n+1}, u, v \leftarrow \mathbb{G}$ and a secret key $x \leftarrow \mathbb{Z}_p$ and compute $y = g_1^x$. Output the secret key $sk = x$ and the public key $pk = (g_1, h_1, \ldots, h_{n+1}, u, v, y)$.
$\mathsf{Sign}(sk, \langle m_1, \ldots, m_n\rangle)$	Pick the random values $r, s \leftarrow \mathbb{Z}_p$ and compute the value $T = (g_1 h_1^{m_1} \cdots h_n^{m_n} h_{n+1}^r)^{1/(x+s)}$. Output the signature $s = (T, r, s)$.
$\mathsf{VfSig}(pk, s, \langle m_1, \ldots, m_n\rangle)$	Parse the public key pk as $(g_1, h_1, \ldots, h_{n+1}, u, v)$ and the signature s as (T, r, s). Check if $e(T, g_1^s y) = e(g_1 h_1^{m_1} \cdots h_n^{m_n} h_{n+1}^r, g_1)$.

To compute a proof of knowledge of signature possession, a prover picks random $t, t' \leftarrow \mathbb{Z}_p$, computes $\hat{T} = Tu^t$ and $Q = v^t u^{t'}$, sends $\hat{T}$ and Q to the verifier, and executes the following proof of knowledge:

$$\mathsf{PK}\Big\{(\alpha, \beta, s, t, t', m_1, \ldots, m_n, r):$$
$$Q = v^t u^{t'} \wedge 1 = Q^{-s} v^\alpha u^\beta \wedge \frac{e(\hat{T}, y)}{e(g_1, g_1)} = e(\hat{T}, g_1)^{-s} \cdot$$
$$\cdot e(u, y)^t e(u, g_1)^\alpha e(h_{n+1}, g_1)^r \prod_{i=1}^{n} e(h_i, g_1)^{m_i}\Big\},$$

where $\alpha = st$ and $\beta = st'$.

This scheme was also employed in the OTAC scheme due to Camenisch et al. [6].

3 Committing Ciphertex-policy attribute-based encryption with key separation

A ciphertex-policy attribute-based encryption (CPABE) scheme [3] possesses the algorithms ($\mathsf{ABESetup}$, ABEExt, ABEEnc, ABEDec). The key generation center ($\mathcal{KGC}$) executes $\mathsf{ABESetup}(1^k)$ to output public parameters par and a master secret key msk. A user $\mathcal{U}$ with a set of attributes $\mathbb{A}$ queries $\mathcal{KGC}$, which runs $\mathsf{ABEExt}(msk, \mathbb{A})$ and returns a secret key $sk_\mathbb{A}$. On input a message m and an access structure $\mathbb{P}$, $\mathsf{ABEEnc}(par, m, \mathbb{P})$ computes a ciphertext ct that can only be decrypted by a user $\mathcal{U}$ that possesses a set of attributes $\mathbb{A}$ that satisfies $\mathbb{P}$ (ct implicitly contains $\mathbb{P}$). Algorithm $\mathsf{ABEDec}(par, ct, sk_\mathbb{A})$, on input a ciphertext ct and a secret key $sk_\mathbb{A}$, outputs the message m if $\mathbb{A}$ satisfies $\mathbb{P}$.

Definition 7 (*Secure CPABE*) Security for CPABE is defined through the following game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

Setup	$\mathcal{C}$ runs $\mathsf{ABESetup}(1^k)$, keeps msk and sends par to $\mathcal{A}$.
Phase 1	$\mathcal{A}$ requests secret keys for sets of attributes $\mathbb{A}_1, \ldots, \mathbb{A}_{q_1}$.

Challenge $\mathcal{A}$ sends two equal length messages m_0 and m_1 and an access structure $\mathbb{P}^*$ such that none of the sets $\mathbb{A}_1, \ldots, \mathbb{A}_{q_1}$ satisfy $\mathbb{P}^*$. $\mathcal{C}$ picks a random bit b, runs $ct^* = \mathsf{ABEEnc}(par, m_b, \mathbb{P}^*)$ and sends ct^* to $\mathcal{A}$.

Phase 2 $\mathcal{A}$ requests secret keys for sets of attributes $\mathbb{A}_{q_1+1}, \ldots, \mathbb{A}_q$ that do not satisfy $\mathbb{P}^*$.

Guess $\mathcal{A}$ outputs a guess b' of b.

A CPABE scheme is secure if, for all p.p.t. $\mathcal{A}$, $|\Pr[b' = b] - \frac{1}{2}|$ is negligible.

We consider CPABE schemes that possess a property we call key separation. In a CPABE with key separation scheme, the secret key $sk_{\mathbb{A}}$ for an attribute set $\mathbb{A} = \{a_1, \ldots, a_L\}$ consists of $(sk_{a_1}, \ldots, sk_{a_L}, sk'_{\mathbb{A}})$, were $sk'_{\mathbb{A}}$ is computed without knowledge of the attributes and each sk_{a_l}, for all $l \in [1, L]$, is computed on input the attribute a_l only. CPABE schemes that fulfill this property include [3,28].

Definition 8 (*Key Separation for CPABE*) The key separation property requires that there exist algorithms $\mathsf{ABEExtInit}$ and $\mathsf{ABEExtAtt}$ such that, for any ppt adversary $\mathcal{A}$:

$$\Pr\left[\begin{array}{l}(par, msk) \leftarrow \mathsf{ABESetup}(1^k);\\ (\mathbb{A}, \text{info}) \leftarrow \mathcal{A}(par);\\ sk_{\mathbb{A}} \leftarrow \mathsf{ABEExt}(msk, \mathbb{A}) :\\ 1 = \mathcal{A}(\text{info}, sk_{\mathbb{A}})\end{array}\right] \approx$$

$$\Pr\left[\begin{array}{l}(par, msk) \leftarrow \mathsf{ABESetup}(1^k);\\ (\mathbb{A}, \text{info}) \leftarrow \mathcal{A}(par);\\ (sk'_{\mathbb{A}}, \text{st}) \leftarrow \mathsf{ABEExtInit}(msk);\\ \forall a \in \mathbb{A} : sk_a \leftarrow \mathsf{ABEExtAtt}(\text{st}, a);\\ sk_{\mathbb{A}} \leftarrow (sk'_{\mathbb{A}}, \{sk_a\}_{a\in\mathbb{A}}) :\\ 1 = \mathcal{A}(\text{info}, sk_{\mathbb{A}})\end{array}\right]$$

For a CPABE scheme to be used in our OTAC schemes, we need two additional properties. First, we require an efficient zero-knowledge proof of knowledge of the statement $\mathsf{PK}\{(msk) : (par, msk) \leftarrow \mathsf{ABESetup}(1^k)\}$. Second, we need it to be committing in the sense defined in [15] for identity-based encryption.

The committing property avoids that, given a ciphertext ct encrypted under an access structure $\mathbb{P}$, two different secret keys associated with two (possibly) different sets of attributes that satisfy $\mathbb{P}$ yield a different result when decrypting ct. In our OTAC construction, this prevents a malicious sender from computing incorrectly formed ciphertexts that decrypt differently depending on the secret key used, which would threaten receiver's security.

We define an algorithm $\mathsf{ABEValidSK}(par, \mathbb{A}, sk_{\mathbb{A}})$ that performs a correctness check on par and $sk_{\mathbb{A}}$ and outputs 1 if the check succeeds and 0 otherwise. We also define a ciphertext validity check algorithm $\mathsf{ABEValidCT}(par, ct)$ that performs a correctness check on par and ct and outputs 1 if the check succeeds and 0 otherwise. These algorithms are useful when the CPABE scheme fulfills the following definition.

Definition 9 (*Committing CPABE*) A CPABE scheme is committing if and only if: (1) it is secure in the sense of Definition 7 and (2) every p.p.t. adversary $\mathcal{A}$ has negligible advantage in k in the following game: First, $\mathcal{A}$ outputs parameters par, a ciphertext ct under access structure $\mathbb{P}$, and two (possibly) different sets of attributes $\mathbb{A}$ and $\mathbb{A}'$ that satisfy $\mathbb{P}$. If $\mathsf{ABEValidCT}(par, ct)$ outputs 0 then abort. Otherwise the challenger runs the ABEExt protocol with $\mathcal{A}$ twice on input $(par, \mathbb{A})$ and $(par, \mathbb{A}')$ to obtain $sk_{\mathbb{A}}$ and $sk_{\mathbb{A}'}$. The challenger runs $\mathsf{ABEValidSK}(par, \mathbb{A}, sk_{\mathbb{A}})$ and $\mathsf{ABEValidSK}(par, \mathbb{A}, sk_{\mathbb{A}'})$ and aborts if any of them outputs 0. $\mathcal{A}$'s advantage is $\Pr[\mathsf{ABEDec}(par, ct, sk_{\mathbb{A}}) \neq \mathsf{ABEDec}(par, ct, sk_{\mathbb{A}'})]$.

4 Ideal functionalities $\mathcal{F}_{\text{REG}}$, $\mathcal{F}_{\text{SMT}}$, $\mathcal{F}_{\text{ASMT}}$, $\mathcal{F}_{\text{CRS}}^{\mathcal{D}}$, $\mathcal{F}_{\text{AA}}$, $\mathcal{F}_{\text{NOT}}$ and $\mathcal{F}_{\text{AOT}}$

4.1 Universally composable security

The universal composability framework [10] is a general framework for analyzing the security of cryptographic protocols in arbitrary composition with other protocols. The security of a protocol φ is analyzed by comparing the view of an environment $\mathcal{Z}$ in a real execution of φ against that of $\mathcal{Z}$ when interacting with an ideal functionality $\mathcal{F}$ that carries out the desired task. The environment $\mathcal{Z}$ chooses the inputs of the parties and collects their outputs. In the real world, $\mathcal{Z}$ can communicate freely with an adversary $\mathcal{A}$ who controls the network as well as any corrupt parties. In the ideal world, $\mathcal{Z}$ interacts with dummy parties, who simply relay inputs and outputs between $\mathcal{Z}$ and $\mathcal{F}$, and a simulator $\mathcal{V}$. We say that a protocol φ securely realizes $\mathcal{F}$ if $\mathcal{Z}$ cannot distinguish the real world from the ideal world, i.e., $\mathcal{Z}$ cannot distinguish whether it is interacting with $\mathcal{A}$ and parties running protocol φ or with $\mathcal{V}$ and dummy parties relaying to $\mathcal{F}_{\varphi}$.

More formally, let $k \in \mathbb{N}$ denote the security parameter and $a \in \{0,1\}^*$ denote an input. Two binary distribution ensembles $X = \{X(k,a)\}_{k\in\mathbb{N},a\in\{0,1\}^*}$ and $Y = \{Y(k,a)\}_{k\in\mathbb{N},a\in\{0,1\}^*}$ are indistinguishable ($X \approx Y$) if for any $c, d \in \mathbb{N}$ there exists $k_0 \in \mathbb{N}$ such that for all $k > k_0$ and all $a \in \cup_{\kappa\leq k^d}\{0,1\}^{\kappa}$, $|\Pr[X(k,a) = 1] - \Pr[Y(k,a) = 1]| < k^{-c}$. Let $\text{REAL}_{\varphi,\mathcal{A},\mathcal{Z}}(k,a)$ denote the random variable given by the output of $\mathcal{Z}$ when executed on input a with $\mathcal{A}$ and parties running φ, and let $\text{IDEAL}_{\mathcal{F},\mathcal{V},\mathcal{Z}}(k,a)$ denote the output distribution of $\mathcal{Z}$ when executed on a with $\mathcal{V}$ and dummy parties relaying to $\mathcal{F}$. We say that protocol φ securely realizes $\mathcal{F}$ if, for all polynomial-time $\mathcal{A}$, there exists a polynomial-time $\mathcal{V}$ such that, for all polynomial-time $\mathcal{Z}$, $\text{REAL}_{\varphi,\mathcal{A},\mathcal{Z}} \approx \text{IDEAL}_{\mathcal{F},\mathcal{V},\mathcal{Z}}$.

When describing ideal functionalities, we use the following conventions:

Aborts	When we say that an ideal functionality $\mathcal{F}$ aborts after being activated with a message $(*,\ldots)$, we mean that $\mathcal{F}$ halts the execution of its program and sends a special abortion message $(*,\perp)$ to the party that invoked the functionality.
Network versus local communication	The identity of an ITM instance (ITI) consists of a party identifier *pid* and a session identifier *sid*. A set of parties in an execution of a system of ITMs are a protocol instance if they have the same session identifier *sid*. ITIs can pass direct inputs to and outputs from "local" ITIs that have the same *pid*. An ideal functionality $\mathcal{F}$ has $pid = \perp$ and is considered local to all parties. An instance of $\mathcal{F}$ with session identifier *sid* only accepts inputs from and passes outputs to machines with the same session identifier *sid*. When describing functionalities, the expressions "output to $\mathcal{P}$" and "on input from $\mathcal{P}$", where $\mathcal{P}$ is a party identity *pid*, mean that the output is passed to and the input is received from party $\mathcal{P}$ only. Communication between ITIs with different party identifiers must take place over the network. The network is controlled by the adversary, meaning that he can arbitrarily delay, modify, drop, or insert messages.
Waiting for the simulator	When we say that $\mathcal{F}$ *sends* m *to* $\mathcal{V}$ *and waits for* m' *from* $\mathcal{V}$, we mean that $\mathcal{F}$ chooses a unique execution identifier, saves its current state, and sends m together with the identifier to $\mathcal{V}$. When $\mathcal{V}$ invokes a dedicated resume interface with a message m' and an execution identifier, $\mathcal{F}$ looks up the execution state associated to the identifier and continues running its program where it left off using m'.

A protocol $\varphi^{\mathcal{G}}$ securely realizes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model when φ is allowed to invoke the ideal functionality $\mathcal{G}$. Therefore, for any protocol ψ that securely realizes functionality $\mathcal{G}$, the composed protocol φ^{ψ}, which is obtained by replacing each invocation of an instance of $\mathcal{G}$ with an invocation of an instance of ψ, securely realizes $\mathcal{F}$.

Our OTAC protocol in Sect. 5 works in a hybrid model that employs, among others, the ideal functionality for anonymous attribute authentication in Sect. 4.3 and the ideal functionality for OT in Sect. 4.4. This construction is UC secure in this hybrid model. The constructions we propose for anonymous attribute authentication in Sect. 6.2 and for OT in Sect. 6.3, for the sake of efficiency, employ the Fiat-Shamir transform to compute zero-knowledge proofs, and thus do not achieve UC security.

4.2 Ideal functionalities $\mathcal{F}_{\text{REG}}$, $\mathcal{F}_{\text{SMT}}$, $\mathcal{F}_{\text{ASMT}}$, $\mathcal{F}_{\text{CRS}}^{\mathcal{D}}$

Our protocol makes use of the functionalities $\mathcal{F}_{\text{REG}}$ for key registration, $\mathcal{F}_{\text{SMT}}$ for secure message transmission, $\mathcal{F}_{\text{ASMT}}$ for anonymous secure message transmission, and $\mathcal{F}_{\text{CRS}}^{\mathcal{D}}$, parameterized by a distribution $\mathcal{D}$, for common reference strings. We take the description of this functionalities from [8]. We modify the description of $\mathcal{F}_{\text{ASMT}}$ to allow the receiver of a message from an anonymous sender to reply to this message. To do this, the functionality $\mathcal{F}_{\text{ASMT}}$ generates a pseudonym for the sender and gives it to the receiver. The receiver can use this pseudonym in order to reply to sender.

Functionality $\mathcal{F}_{\text{REG}}$

1. On input (reg.register, sid, v) from a party $\mathcal{P}$:
   - If $sid \neq (\mathcal{P}, sid')$ or there is a value v stored, exit the program.
   - Store v.
   - Send (reg.register, sid, v) to $\mathcal{V}$ and wait for (reg.register, sid) from $\mathcal{V}$.
   - Send (reg.register, sid) to $\mathcal{P}$.
2. On input (reg.retrieve, sid) from any party $\mathcal{P}'$:
   - If there is a value v stored, set $v' = v$, else set $v' = \perp$.
   - Send (reg.retrieve, sid, v') to $\mathcal{V}$ and wait for (reg.retrieve, sid) from $\mathcal{V}$.
   - Send (reg.retrieve.end, sid, v') to $\mathcal{P}'$.

Functionality $\mathcal{F}_{\text{SMT}}$

Parameterized by a leakage function $l : \{0, 1\}^* \rightarrow \{0, 1\}^*$, $\mathcal{F}_{\text{SMT}}$ works as follows:

1. On input (smt.send, sid, m) from a party $\mathcal{T}$, execute the following program:
   - If $sid \neq (\mathcal{T}, \mathcal{R}, sid')$, exit the program.
   - Send (smt.send, sid, $l(m)$) to $\mathcal{V}$.
   - Wait for a message (smt.send, sid) from $\mathcal{V}$.
   - Send (smt.send.end, sid, m) to $\mathcal{R}$.

Functionality $\mathcal{F}_{\text{ASMT}}$

Parameterized by a leakage function $l : \{0, 1\}^* \rightarrow \{0, 1\}^*$, $\mathcal{F}_{\text{ASMT}}$ works as follows:

1. On input (asmt.send, sid, m) from a party $\mathcal{T}$, execute the following program:
   - If $sid \neq (\mathcal{R}, sid')$, exit the program.
   - Send (asmt.send, sid', $l(m)$) to $\mathcal{V}$ and wait for a message (asmt.send, sid') from $\mathcal{V}$.
   - Generate a random pseudonym P and store a tuple $(\mathcal{T}, P, \mathcal{R})$.
   - Send (asmt.send.end, sid, m, P) to $\mathcal{R}$.

2. On input (asmt.reply, sid, m, P) from $\mathcal{R}$, execute the following program:
   - If there is not a tuple ($\mathcal{T}$, P, $\mathcal{R}$) stored, exit the program.
   - Send (asmt.reply, sid', $l(m)$) to $\mathcal{V}$ and wait for a message (asmt.reply, sid') from $\mathcal{V}$.
   - Send (asmt.reply.end, sid, m) to $\mathcal{T}$.

Functionality $\mathcal{F}_{\text{CRS}}^{\mathcal{D}}$

Parameterized with a distribution $\mathcal{D}$, $\mathcal{F}_{\text{CRS}}$ works as follows:

1. On input (crs.get, sid) from party $\mathcal{P}$, execute the following program:
   - If there is no value r recorded, pick $r \leftarrow \mathcal{D}$ and store r.
   - Send (crs.get, sid, r) to $\mathcal{V}$ and wait for (crs.get, sid) from $\mathcal{V}$.
   - Send (crs.get.end, sid, r) to $\mathcal{P}$.

4.3 Ideal functionality of anonymous attribute authentication $\mathcal{F}_{\text{AA}}$

We depict the ideal functionality of anonymous attribute authentication $\mathcal{F}_{\text{AA}}$. This functionality interacts with an issuer $\mathcal{I}$ and users $\mathcal{U}$.

The issuer $\mathcal{I}$ issues some attributes $\langle a_l \rangle_{l=1}^{L}$ to a user $\mathcal{U}$. A user $\mathcal{U}$ can prove to any party that a set of commitments $\langle com_l \rangle_{l=1}^{L}$ commit to attributes certified by $\mathcal{I}$.

The interaction between the functionality $\mathcal{F}_{\text{AA}}$ and the issuer $\mathcal{I}$ and the users $\mathcal{U}$ takes place through the interfaces aa.setup, aa.getparams, aa.issue and aa.prove.

1. The issuer $\mathcal{I}$ uses the aa.setup interface to send the parameters par_c and the verification algorithm VfCom of a commitment scheme.
2. Any party $\mathcal{P}$ employs the aa.getparams interface to obtain the parameters par_c and the verification algorithm VfCom.
3. The issuer aa.issue uses the aa.issue interface to issue some attributes $\langle a_l \rangle_{l=1}^{L}$ to a user $\mathcal{U}$.
4. A user $\mathcal{U}$ employs the aa.prove interface to prove to any party $\mathcal{P}$ that the commitments $\langle com_l \rangle_{l=1}^{L}$ commit to attributes $\langle a_l \rangle_{l=1}^{L}$ that were issued by the issuer $\mathcal{I}$ to the user $\mathcal{U}$. The party $\mathcal{P}$ receives the commitments $\langle com_l \rangle_{l=1}^{L}$.

$\mathcal{F}_{\text{AA}}$ employs a table Tbl_a. Tbl_a stores entries of the form $[\mathcal{U}, \mathbb{A}]$, which map a user $\mathcal{U}$ to a set of attributes $\mathbb{A}$ issued by $\mathcal{I}$.

The use of commitments in the functionality $\mathcal{F}_{\text{AA}}$ allows $\mathcal{F}_{\text{AA}}$ to be used along with other functionalities in a hybrid protocol in which it is necessary to guarantee that the inputs to the functionalities are equal. For example, in our OTAC constructions in Sect. 5.1, $\mathcal{F}_{\text{AA}}$ is used along with $\mathcal{F}_{\text{NOT}}$. The use of commitments both in $\mathcal{F}_{\text{AA}}$ and in $\mathcal{F}_{\text{NOT}}$ allows the verifying party $\mathcal{P}$ to check that the attributes input to $\mathcal{F}_{\text{NOT}}$ are equal to those input to $\mathcal{F}_{\text{AA}}$ and thus were issued by the issuer.

Functionality $\mathcal{F}_{\text{AA}}$

Parameterized with a maximum number of attributes L_{max} and a universe of attributes Ψ, and running with a credential issuer $\mathcal{I}$ and users $\mathcal{U}$, $\mathcal{F}_{\text{AA}}$ works as follows:

1. On input (aa.setup, sid, par_c, VfCom) from $\mathcal{I}$, do the following:
   - Abort if $sid \neq (\mathcal{I}, sid')$ or if (par_c, VfCom) is already stored.
   - Send (aa.setup, sid, par_c, VfCom) to $\mathcal{V}$ and wait for a message (aa.setup, sid) from $\mathcal{V}$.
   - Store (par_c, VfCom) and initialize an empty table Tbl_a.
   - Output (aa.setup.end, sid) to $\mathcal{I}$.
2. On input (aa.getparams, sid) from any party $\mathcal{P}$:
   - Abort if there is no tuple (par_c, VfCom) stored.
   - Send (aa.getparams, sid, par_c, VfCom) to $\mathcal{V}$ and wait for (aa.getparams, sid) from $\mathcal{V}$.
   - Send (aa.getparams, sid, par_c, VfCom) to $\mathcal{P}$.
3. On input (aa.issue, $sid, \mathcal{U}, \langle a_l \rangle_{l=1}^{L}$) from $\mathcal{I}$, do the following:
   - Abort if there is no tuple (par_c, VfCom) stored, or if $L > L_{max}$, or if for any $l \in [1, L]$, $a_l \notin \Psi$.
   - Send (aa.issue, $sid, \mathcal{U}$) to $\mathcal{V}$ and wait for (aa.issue, sid) from $\mathcal{V}$.
   - Set $\mathbb{A} = \{\{a_1\} \cup \ldots \cup \{a_L\}\}$. If $\mathcal{U}$ is honest then append $[\mathcal{U}, \mathbb{A}]$ to Tbl_a, else append $[\mathcal{V}, \mathbb{A}]$ to Tbl_a.
   - Send (aa.issue.end, sid, $\langle a_l \rangle_{l=1}^{L}$) to $\mathcal{U}$.
4. On input (aa.prove, sid, $\mathcal{P}$, $\langle com_l, a_l, open_l \rangle_{l=1}^{L}$) from a user $\mathcal{U}$:
   - Abort if there is no tuple (par_c, VfCom) stored, or if $1 \neq \mathsf{VfCom}(par_c, com_l, a_l, open_l)$ for any $l \in [1, L]$, or if $\mathcal{I}$ or $\mathcal{U}$ are honest and there is no entry $[\mathcal{U}', \mathbb{A}]$ in Tbl_a such that $\langle a_l \rangle_{l=1}^{L} \subseteq \mathbb{A}$, where $\mathcal{U}' = \mathcal{U}$ if $\mathcal{U}$ is honest or $\mathcal{U}' = \mathcal{V}$ if $\mathcal{U}$ is corrupt.
   - Send (aa.prove.end, sid, $\mathcal{P}$) to $\mathcal{V}$ and wait for (aa.prove.end, sid) from $\mathcal{V}$.
   - Send (aa.prove.end, sid, $\langle com_l \rangle_{l=1}^{L}$) to $\mathcal{P}$.

We discuss the interfaces of $\mathcal{F}_{\text{AA}}$.

1. The aa.setup interface is invoked by the issuer $\mathcal{I}$ on input the commitment parameters par_c and a commitment verification algorithm VfCom. This allows users of $\mathcal{F}_{\text{AA}}$ to prove that the attributes certified by $\mathcal{I}$ are committed to in commitments generated outside the functionality. $\mathcal{F}_{\text{AA}}$ sends par_c and VfCom to the simulator $\mathcal{V}$ and stores them. The restriction that the issuer's identity must be included in the session identifier $sid = (\mathcal{I}, sid')$ guarantees that each issuer can initialize its own instance of the functionality.
2. The aa.getparams interface allows any party to request par_c and VfCom.
3. The aa.issue interface is called by the issuer on input a user identity and some attributes. The simulator indicates when the issuance is to be finalized by sending a (aa.issue, sid) message. At this point, the issuance is recorded in Tbl_a. If the user is honest, the issuance is recorded under the correct user's identity, which requires any instantiating protocol to set up an authenticated channel to the user to ensure this in the real world. If the user is corrupt, the attribute is recorded as belonging to the simulator,

modeling that corrupt users may pool their credentials. Note that the simulator is not given the issued attribute values, so the real-world protocol must hide these from the adversary.

4. The aa.prove interface lets a user $\mathcal{U}$ prove to any party $\mathcal{P}$ that the attributes $\langle a_l \rangle_{l=1}^{L}$ issued to her by $\mathcal{I}$ are committed in the commitments $\langle com_l \rangle_{l=1}^{L}$. An honest user can only show a set of attributes that was issued to her. If the issuer is honest, but the user is corrupt, then the presented attributes could be issued to any corrupt user. Note that neither the verifying party nor the simulator learns the identity of the user who initiated the proof protocol. One requires some form of anonymous communication between the user and the verifying party to achieve this in the real world protocol.

4.4 Ideal functionality of oblivious transfer $\mathcal{F}_{\text{NOT}}$

We depict the ideal functionality of non-adaptive OT $\mathcal{F}_{\text{NOT}}$. This functionality interacts with a sender $\mathcal{E}$ and a receiver $\mathcal{R}$.

The sender $\mathcal{E}$ possesses the messages $(m_1, \ldots, m_N)$. The receiver $\mathcal{R}$ chooses the selection values $\langle \sigma_k \rangle_{k=1}^{K}$, where each $\sigma_k \in [1, N]$, and obtains the messages $\langle m_{\sigma_k} \rangle_{k=1}^{K}$.

The interaction between the functionality $\mathcal{F}_{\text{NOT}}$ and the sender $\mathcal{E}$ and the receiver $\mathcal{R}$ takes place through the interfaces not.initrec, not.init, not.request and not.transfer.

1. The receiver $\mathcal{R}$ sends the sender identity $\mathcal{E}$ to the functionality $\mathcal{F}_{\text{NOT}}$ through the not.initrec interface. We note that the receiver $\mathcal{R}$ remains anonymous towards the sender $\mathcal{E}$.
2. The sender $\mathcal{E}$ uses the not.init interface to send the messages $(m_1, \ldots, m_N)$ to the functionality $\mathcal{F}_{\text{NOT}}$ along with the parameters par_c and the verification algorithm VfCom of a commitment scheme.
3. The receiver $\mathcal{R}$ employs the not.request interface to request the message associated with the selection values $\langle \sigma_k \rangle_{k=1}^{K}$. $\mathcal{R}$ also sends commitments to each σ_k and the openings of these commitments. The sender $\mathcal{E}$ obtains the commitments.
4. The sender $\mathcal{E}$ employs the not.transfer interface to transfer to the receiver the messages associated to the selection values that were committed.

There are two main differences between $\mathcal{F}_{\text{NOT}}$ and previous functionalities for non-adaptive OT [16]. First, in our functionality the sender obtains from the receiver commitments to the selection values $\langle \sigma_k \rangle_{k=1}^{K}$. This feature allows $\mathcal{F}_{\text{NOT}}$ to be used in a hybrid protocol that also employs $\mathcal{F}_{\text{AA}}$, like our OT with access control protocol in Sect. 5.2. The commitments are employed to guarantee that some inputs to $\mathcal{F}_{\text{NOT}}$ are equal to some inputs to $\mathcal{F}_{\text{AA}}$. For this purpose, commitments to the inputs are computed and sent both to $\mathcal{F}_{\text{NOT}}$ and to $\mathcal{F}_{\text{AA}}$.

Second, $\mathcal{F}_{\text{NOT}}$ does not limit to a maximum $K < N$ the number of messages that can be transferred. The reason for the latter is that this maximum, when the functionality is employed as building block of our OT with access control protocol, is determined by the number of attributes a receiver possesses.

Functionality $\mathcal{F}_{\text{NOT}}$

Parameterized with a number of messages N and a message space $\mathcal{M}$, and running with a sender $\mathcal{E}$ and a receiver $\mathcal{R}$, $\mathcal{F}_{\text{NOT}}$ works as follows:

1. On input (not.initrec, *sid*) from $\mathcal{R}$, $\mathcal{F}_{\text{NOT}}$ does the following:
   - Abort if $sid \neq (\mathcal{E}, sid')$ or if init already exists.
   - Send (not.initrec, *sid*) to $\mathcal{V}$ and wait for (not.initrec, *sid*) from $\mathcal{V}$.
   - Create init.
   - Send (not.initrec.end, *sid*) to $\mathcal{E}$.
2. On input (not.init, $sid, m_1, \ldots, m_N, par_c$, VfCom) from $\mathcal{E}$, $\mathcal{F}_{\text{NOT}}$ does the following:
   - Abort if init does not exist, or if $(m_1, \ldots, m_N, par_c, \text{VfCom})$ is already stored, or if, for $i = 1$ to N, $m_i \notin \mathcal{M}$.
   - Send (not.init, sid, par_c, VfCom) to $\mathcal{V}$ and wait for (not.init, *sid*) from $\mathcal{V}$.
   - Store $(m_1, \ldots, m_N, par_c, \text{VfCom})$.
   - Send (not.init.end, sid, par_c, VfCom) to $\mathcal{R}$.
3. On input (not.request, $sid, \langle com_k, \sigma_k, open_k \rangle_{k=1}^K$) from $\mathcal{R}$, $\mathcal{F}_{\text{NOT}}$ does the following:
   - Abort if $(m_1, \ldots, m_N, par_c, \text{VfCom})$ is not stored, or if there is a tuple $\langle com'_k, \sigma'_k, open'_k \rangle_{k=1}^K$ already stored, or if $1 \neq \text{VfCom}(par_c, com_k, \sigma_k, open_k)$ or $\sigma_k \notin [1, N]$ for any $k \in [1, K]$.
   - Send (not.request, *sid*) to $\mathcal{V}$ and wait for a message (not.request, *sid*) from $\mathcal{V}$.
   - Store $\langle com_k, \sigma_k, open_k \rangle_{k=1}^K$.
   - Send (not.request.end, $sid, \langle com_k \rangle_{k=1}^K$) to $\mathcal{E}$.
4. On input (not.transfer, $sid, \langle com_k \rangle_{k=1}^K$) from $\mathcal{R}$, $\mathcal{F}_{\text{NOT}}$ does the following:
   - Abort if $\langle com'_k, \sigma_k, open_k \rangle_{k=1}^K$ such that $\langle com'_k = com_k \rangle_{k=1}^K$ is not stored.
   - Send (not.transfer, *sid*) to $\mathcal{V}$ and wait for a message (not.transfer, *sid*) from $\mathcal{V}$.
   - Send (not.transfer.end, $sid, \langle \sigma_k, m_{\sigma_k} \rangle_{k=1}^K$) to $\mathcal{R}$.

We describe the interfaces of $\mathcal{F}_{\text{NOT}}$.

1. The not.initrec interface is invoked by the receiver $\mathcal{R}$ in order to send to the functionality the identity of the sender. The functionality creates a variable init to indicate the receiver is ready and informs the sender $\mathcal{E}$.
2. The sender $\mathcal{E}$ invokes not.init on input the messages $(m_1, \ldots, m_N)$, the parameters of the commitment scheme par_c and the commitment verification algorithm VfCom. The functionality aborts if the receiver is not ready, or if the initialization phase has already been run. Then the functionality communicates to the simulator that the initialization takes place, and the simulator allows the functionality to end the initialization phase. Note that the simulator learns par_c and VfCom but does not learn the messages $(m_1, \ldots, m_N)$. Finally, the functionality sends par_c and VfCom to the receiver.
3. The receiver invokes not.request on input K commitments to selection values σ with openings *open*. $\mathcal{F}_{\text{NOT}}$ aborts if the initialization phase has not been run, if a request was already received, or if the openings of the commitments are not correct. Then the simulator indicates to the functionality that the request phase can be finalized. $\mathcal{F}_{\text{NOT}}$ stores

the commitments, the selection values and the openings and sends the commitments to the sender.
4. The sender invokes not.transfer on input K commitments. $\mathcal{F}_{\text{NOT}}$ aborts if those commitments were not sent before by the receiver. Then the simulator indicates to the functionality that the transfer phase can be finalized. Finally, the functionality sends the selected messages to the receiver.

4.5 Ideal functionality of adaptive oblivious transfer $\mathcal{F}_{\text{AOT}}$

We depict the ideal functionality of adaptive OT $\mathcal{F}_{\text{AOT}}$ [9,16]. This functionality interacts with a sender $\mathcal{E}$ and receivers $\mathcal{R}$.

The sender $\mathcal{E}$ possesses the messages $(m_1, \ldots, m_N)$. There are more than one transfer phases. At each transfer phase, a receiver $\mathcal{R}$ chooses a selection value $\sigma \in [1, N]$ and obtains the message m_σ. The difference with respect to the non-adaptive case is that the receiver can choose a selection value after obtaining the messages from previous transfer phases.

The interaction between the functionality $\mathcal{F}_{\text{AOT}}$ and the sender $\mathcal{E}$ and the receivers $\mathcal{R}$ takes place through the interfaces aot.init and aot.transfer.

1. The sender $\mathcal{E}$ uses the aot.init interface to send the messages $(m_1, \ldots, m_N)$ to the functionality $\mathcal{F}_{\text{AOT}}$.
2. A receiver $\mathcal{R}$ uses the aot.transfer interface on input a selection value σ in order to obtain the message m_σ.

$\mathcal{F}_{\text{AOT}}$, unlike the functionalities in [9,16], does not limit to a maximum $K < N$ the number of messages that can be transferred. The reason for the latter is that this maximum, when the functionality is employed as building block of our OT with access control protocol, is determined by the number of messages whose access control policy is fulfilled by the receiver's attributes. A second difference between $\mathcal{F}_{\text{AOT}}$ and the functionalities in [9,16] is that $\mathcal{F}_{\text{AOT}}$ interacts with more than one receiver and receivers remain anonymous towards the sender.

Functionality $\mathcal{F}_{\text{AOT}}$

Parameterized with a number of messages N and a message space $\mathcal{M}$, and running with a sender $\mathcal{E}$ and receivers $\mathcal{R}$, $\mathcal{F}_{\text{AOT}}$ works as follows:

1. On input (aot.init, $sid, m_1, \ldots, m_N$) from $\mathcal{E}$, $\mathcal{F}_{\text{AOT}}$ does the following:
   - Abort if $sid \neq (\mathcal{E}, sid')$, or if $(m_1, \ldots, m_N)$ is already stored, or if, for $i = 1$ to N, $m_i \notin \mathcal{M}$.
   - Send (aot.init, sid) to $\mathcal{V}$ and wait for (aot.init, sid) from $\mathcal{V}$.
   - Store $(m_1, \ldots, m_N)$.
   - Send (aot.init.end, sid) to $\mathcal{E}$.
2. On input (aot.transfer, sid, σ) from $\mathcal{R}$, $\mathcal{F}_{\text{AOT}}$ does the following:
   - Abort if $(m_1, \ldots, m_N)$ is not stored.
   - Send (aot.transfer, sid) to $\mathcal{V}$.
   - If $\mathcal{E}$ is corrupt, wait for a message (aot.transfer, sid, b) from $\mathcal{V}$, else wait for a message (aot.transfer, sid) from $\mathcal{V}$.
   - If $\mathcal{E}$ is corrupt and $b = 0$, send (aot.transfer.end, $sid, \perp$) to $\mathcal{R}$, else send (aot.transfer.end, sid, m_σ) to $\mathcal{R}$.

5 Oblivious transfer with access control

5.1 Ideal functionalities of oblivious transfer with access control

We depict the ideal functionalities of non-restricted and restricted OTAC, which we denote by $\mathcal{F}_{\text{NOTAC}}$ and $\mathcal{F}_{\text{ROTAC}}$. These functionalities interact with a sender $\mathcal{E}$, an issuer $\mathcal{I}$, and receivers $\mathcal{R}$.

The sender $\mathcal{E}$ possesses a list of messages $(m_1, \ldots, m_N)$. These messages are associated with the access control policies $(\mathbb{P}_1, \ldots, \mathbb{P}_N)$. An access control policy describes the attributes that a receiver must possess in order to be allowed to obtain a message. The attributes that a receiver possesses are issued by $\mathcal{I}$.

The interaction between the functionalities $\mathcal{F}_{\text{NOTAC}}$ and $\mathcal{F}_{\text{ROTAC}}$ and the sender $\mathcal{E}$, the issuer $\mathcal{I}$, and the receivers $\mathcal{R}$ takes place through the interfaces otac.init, otac.getpolicy, otac.issue and notac.transfer or aotac.transfer.

The interfaces otac.init, otac.getpolicy and otac.issue are common for both the functionalities $\mathcal{F}_{\text{NOTAC}}$ and $\mathcal{F}_{\text{ROTAC}}$. Through the initialization interface otac.init, the sender $\mathcal{E}$ sends the messages $(m_1, \ldots, m_N)$ and the policies $(\mathbb{P}_1, \ldots, \mathbb{P}_N)$ to the functionality. The receivers employ the otac.getpolicy interface in order to obtain the policies $(\mathbb{P}_1, \ldots, \mathbb{P}_N)$. Through the issuing interface otac.issue, the issuer issues a list of attributes $\langle a_l \rangle_{l=1}^{L}$ to a receiver $\mathcal{R}$.

The transfer interface is different in $\mathcal{F}_{\text{NOTAC}}$ and in $\mathcal{F}_{\text{ROTAC}}$. In $\mathcal{F}_{\text{NOTAC}}$, the transfer interface notac.transfer allows the receiver $\mathcal{R}$ to send to $\mathcal{F}_{\text{NOTAC}}$ a set of attributes and, if those attributes were issued by the issuer $\mathcal{I}$ to the receiver $\mathcal{R}$, the receiver $\mathcal{R}$ receives from $\mathcal{F}_{\text{NOTAC}}$ all the messages whose access control policy is fulfilled by the set of attributes.

In $\mathcal{F}_{\text{ROTAC}}$, the transfer interface aotac.transfer allows the receiver $\mathcal{R}$ to send to $\mathcal{F}_{\text{ROTAC}}$ a set of attributes and a selection value $\sigma \in [1, N]$. If the attributes in the set were issued by the issuer $\mathcal{I}$ to the receiver $\mathcal{R}$ and fulfill the policy $\mathbb{P}_\sigma$ associated with the message m_σ, then the receiver $\mathcal{R}$ obtains the message m_σ.

$\mathcal{F}_{\text{NOTAC}}$ and $\mathcal{F}_{\text{ROTAC}}$ employ a table Tbl_a. Tbl_a stores entries of the form $[\mathcal{R}, \mathbb{A}]$, which map a receiver $\mathcal{R}$ to a set of attributes $\mathbb{A}$ issued by $\mathcal{I}$. $\mathcal{F}_{\text{NOTAC}}$ and $\mathcal{F}_{\text{ROTAC}}$ also employ a table Tbl_c that stores entries of the form $[\mathcal{R}, \mathbb{A}]$, which store a set of attributes $\mathbb{A}$ for which $\mathcal{R}$ already received the transfer results.

Below we describe formally the functionalities $\mathcal{F}_{\text{NOTAC}}$ and $\mathcal{F}_{\text{ROTAC}}$. After that, we describe more in detail how the interfaces otac.init, otac.getpolicy, otac.issue, notac.transfer and aotac.transfer work.

Functionality $\mathcal{F}_{\text{NOTAC}}$ and $\mathcal{F}_{\text{ROTAC}}$

Parameterized with a number of messages N, a message space $\mathcal{M}$, a maximum number of attributes L_{max} and a universe of attributes Ψ, and running with an issuer $\mathcal{I}$, a sender $\mathcal{E}$ and receivers $\mathcal{R}$, $\mathcal{F}_{\text{NOTAC}}$ and $\mathcal{F}_{\text{ROTAC}}$ work as follows:

1. On input (otac.init, sid, $m_1, \mathbb{P}_1, \ldots, m_N, \mathbb{P}_N$) from $\mathcal{E}$:
   - Abort if $sid \neq (\mathcal{E}, \mathcal{I}, sid')$, or if $(m_1, \mathbb{P}_1, \ldots, m_N, \mathbb{P}_N)$ is already stored, or if, for $i = 1$ to N, $m_i \notin \mathcal{M}$.
   - Send (otac.init, sid, $\mathbb{P}_1, \ldots, \mathbb{P}_N$) to $\mathcal{V}$ and wait for (otac.init, sid) from $\mathcal{V}$.
   - Store $(m_1, \mathbb{P}_1, \ldots, m_N, \mathbb{P}_N)$.
   - Send (otac.init.end, sid) to $\mathcal{E}$.

2. On input (otac.getpolicy, *sid*) from a receiver $\mathcal{R}$:
   - Abort if $(m_1, \mathbb{P}_1, \ldots, m_N, \mathbb{P}_N)$ is not stored.
   - Send (otac.getpolicy, *sid*, $\mathbb{P}_1, \ldots, \mathbb{P}_N$) to $\mathcal{V}$ and receive (otac.getpolicy, *sid*) from $\mathcal{V}$.
   - Send (otac.getpolicy.end, *sid*, $\mathbb{P}_1, \ldots, \mathbb{P}_N$) to $\mathcal{R}$.
3. On input (otac.issue, *sid*, $\mathcal{R}$, $\langle a_l \rangle_{l=1}^{L}$) from $\mathcal{I}$:
   - Abort if $L > L_{max}$ or if $\langle a_l \rangle_{l=1}^{L} \nsubseteq \Psi$.
   - Send (otac.issue, *sid*, $\mathcal{R}$) to $\mathcal{V}$ and wait for (otac.issue, *sid*) from $\mathcal{V}$.
   - Set $\mathbb{A} = \{a_1, \ldots, a_L\}$. If $\mathcal{R}$ is honest then append $[\mathcal{R}, \mathbb{A}]$ to Tbl_a, else append $[\mathcal{V}, \mathbb{A}]$ to Tbl_a.
   - Send (otac.issue.end, *sid*, $\langle a_l \rangle_{l=1}^{L}$) to $\mathcal{R}$.
4. On input (notac.transfer, *sid*, $\mathbb{A}$) from $\mathcal{R}$, $\mathcal{F}_{\text{NOTAC}}$ does the following:
   - Abort if $(m_1, \mathbb{P}_1, \ldots, m_N, \mathbb{P}_N)$ is not stored, or if $\mathcal{I}$ or $\mathcal{R}$ are honest and there is no entry $[\mathcal{R}', \mathbb{A}']$ in Tbl_a such that $\mathbb{A} \subseteq \mathbb{A}'$ and $\mathcal{R}' = \mathcal{R}$ if $\mathcal{R}$ is honest or $\mathcal{R} = \mathcal{V}$ if $\mathcal{R}$ is corrupt.
   - If there is an entry $[\mathcal{R}', \mathbb{A}']$ in Tbl_c such that $\mathbb{A} \subseteq \mathbb{A}'$ and $\mathcal{R}' = \mathcal{R}$ if $\mathcal{R}$ is honest or $\mathcal{R} = \mathcal{V}$ if $\mathcal{R}$ is corrupt, do the following:
     - For $k = 1$ to N, if $\mathbb{A}$ does not satisfy $\mathbb{P}_k$, set $m'_k = \perp$, else set $m'_k = m_k$.
     - Send (notac.transfer.end, *sid*, $\langle m'_k \rangle_{k=1}^{N}$) to $\mathcal{R}$.
   - Else, send (notac.transfer, *sid*, $|\mathbb{A}|$) to $\mathcal{V}$.
   - If $\mathcal{E}$ is corrupt, wait for a message (notac.transfer, *sid*, *b*) from $\mathcal{V}$, else wait for a message (notac.transfer, *sid*) from $\mathcal{V}$.
   - If $\mathcal{E}$ is honest or if $b = 1$, store $[\mathcal{R}, \mathbb{A}]$ in Tbl_c.
   - For $k = 1$ to N, if $\mathcal{E}$ is corrupt and $b = 0$ or if $\mathbb{A}$ does not satisfy $\mathbb{P}_k$, set $m'_k = \perp$, else set $m'_k = m_k$.
   - Send (notac.transfer.end, *sid*, $\langle m'_k \rangle_{k=1}^{N}$) to $\mathcal{R}$.
4. On input (aotac.transfer, *sid*, $\mathbb{A}$, σ) from $\mathcal{R}$, $\mathcal{F}_{\text{ROTAC}}$ does the following:
   - Abort if $(m_1, \mathbb{P}_1, \ldots, m_N, \mathbb{P}_N)$ is not stored, or if $\mathcal{I}$ or $\mathcal{R}$ are honest and there is no entry $[\mathcal{R}', \mathbb{A}']$ in Tbl_a such that $\mathbb{A} \subseteq \mathbb{A}'$ and $\mathcal{R}' = \mathcal{R}$ if $\mathcal{R}$ is honest or $\mathcal{R} = \mathcal{V}$ if $\mathcal{R}$ is corrupt, or if $\sigma \notin [1, N]$.
   - If there is an entry $[\mathcal{R}', \mathbb{A}']$ in Tbl_c such that $\mathbb{A} \subseteq \mathbb{A}'$ and $\mathcal{R}' = \mathcal{R}$ if $\mathcal{R}$ is honest or $\mathcal{R} = \mathcal{V}$ if $\mathcal{R}$ is corrupt, send (aotac.transfer, *sid*, 0) to $\mathcal{V}$, else send (aotac.transfer, *sid*, $|\mathbb{A}|$) to $\mathcal{V}$.
   - If $\mathcal{E}$ is corrupt, wait for a message (aotac.transfer, *sid*, *b*) from $\mathcal{V}$, else wait for a message (aotac.transfer, *sid*) from $\mathcal{V}$.
   - If $\mathcal{E}$ is corrupt and $b = 0$ or if $\mathbb{A}$ does not satisfy $\mathbb{P}_\sigma$, set $m'_\sigma = \perp$, else set $m'_\sigma = m_\sigma$.
   - If $\mathcal{E}$ is honest or if $b = 1$, store $[\mathcal{R}, \mathbb{A}]$ in Tbl_c.
   - Send (aotac.transfer.end, *sid*, m'_σ) to $\mathcal{R}$.

We describe the interfaces of $\mathcal{F}_{\text{NOTAC}}$ and $\mathcal{F}_{\text{ROTAC}}$.

1. The otac.init interface is common for both $\mathcal{F}_{\text{NOTAC}}$ and $\mathcal{F}_{\text{ROTAC}}$. The sender $\mathcal{E}$ invokes otac.init on input the messages and the policies $(m_1, \mathbb{P}_1, \ldots, m_N, \mathbb{P}_N)$. The functionality

aborts if the session identifier does not include the identities of the sender and the issuer. This allows each sender to create his own instance of the functionality and to indicate the identity of the issuer that the sender trusts. The functionality also aborts if the initialization phase has already been run. Then the functionality communicates to the simulator that the initialization phase takes place, and the simulator allows the functionality to end the initialization phase.

2. The otac.getpolicy interface is invoked by a receiver $\mathcal{R}$. If the initialization phase has already been run, the functionality sends the policies to the receiver $\mathcal{R}$.
3. The otac.issue interface is called by the issuer on input a receiver identity $\mathcal{R}$ and some attributes. The simulator indicates when the issuance is to be finalized by sending a (otac.issue, *sid*) message. At this point, the issuance is recorded in Tbl_a. If the receiver is honest, the issuance is recorded under the correct receiver identity $\mathcal{R}$, which requires any instantiating protocol to set up an authenticated channel to the receiver to ensure this in the real world. If the receiver is corrupt, the attribute is recorded as belonging to the simulator $\mathcal{V}$, modeling that corrupt receivers may pool their credentials. Note that the simulator is not given the issued attribute values, so the real-world protocol must hide these from the adversary.
4. (i) In $\mathcal{F}_{\text{NOTAC}}$, the receiver $\mathcal{R}$ invokes notac.transfer on input a set of attributes $\mathbb{A}$. $\mathcal{F}_{\text{NOTAC}}$ aborts if the initialization phase has not been run or if the issuer or the receiver are honest and the receiver was not issued those attributes. If a transfer phase for a set of attributes that includes $\mathbb{A}$ was successfully finalized before, then $\mathcal{F}_{\text{NOTAC}}$ finalizes the transfer phase without contacting the simulator. Else, if the sender is corrupt, $\mathcal{F}_{\text{NOTAC}}$ allows the simulator to input a bit b, which indicates whether the transfer phase should be completed or not. If the sender is honest, $\mathcal{F}_{\text{NOTAC}}$ only asks the simulator to indicate when the transfer phase should be finalized. Finally, the functionality sends to the receiver those messages whose access control policy is satisfied by the attributes in the set.
(ii) In $\mathcal{F}_{\text{ROTAC}}$, the receiver $\mathcal{R}$ invokes aotac.transfer on input a set of attributes and a selection value. $\mathcal{F}_{\text{ROTAC}}$ aborts if the initialization phase has not been run, if the issuer or the receiver are honest and $\mathcal{R}$ was not issued those attributes, or if the selection value is not valid. If a transfer phase for a set of attributes that includes $\mathbb{A}$ was successfully finalized before, $\mathcal{F}_{\text{ROTAC}}$ invokes the simulator on input 0, else on input the attribute size. This indicates to the simulator whether the receiver inputs a set of attributes that was already input in a previous transfer phase or not. The simulator replies as in $\mathcal{F}_{\text{NOTAC}}$. Finally, $\mathcal{F}_{\text{ROTAC}}$ sends the receiver $\mathcal{R}$ the requested message if its associated access control policy is satisfied by the set of attributes that was input by the receiver.

5.2 Constructions of oblivious transfer with access control

We depict our construction of non-restricted and restricted OTAC, which we denote by NOTAC and ROTAC respectively. Both the constructions NOTAC and ROTAC employ a commitment scheme (CSetup, Com and VfCom) as described in Sect. 2.4 and a committing ciphertext-policy attribute-based encryption (CPABE) with key separation scheme with algorithms (ABESetup, ABEExt, ABEEnc, ABEDec, ABEValidSK, ABEValidCT) as described in Sect. 3. They also employ a zero-knowledge proof of knowledge scheme $\mathsf{PK}\{(msk) : (par, msk) \leftarrow \mathsf{ABESetup}(1^k)\}$, i.e., a zero-knowledge proof of knowledge of the master secret key *msk* of the CPABE scheme. Additionally, the constructions NOTAC and ROTAC employ a hash function H, which is modeled as a random oracle.

The constructions NOTAC and ROTAC use the $\mathcal{F}_{\text{CRS}}^{\text{CSetup}}$ and $\mathcal{F}_{\text{REG}}$-hybrid model, where parties make use of the ideal functionalities $\mathcal{F}_{\text{CRS}}^{\text{CSetup}}$ and $\mathcal{F}_{\text{REG}}$ described in Sect. 4.2. $\mathcal{F}_{\text{CRS}}^{\text{CSetup}}$ is parameterized by the commitment setup algorithm CSetup. Additionally, the constructions NOTAC and ROTAC also employ the anonymous attribute authentication functionality $\mathcal{F}_{\text{AA}}$ described in Sect. 4.3 and the non-adaptive OT functionality $\mathcal{F}_{\text{NOT}}$ described in Sect. 4.4. The construction ROTAC also uses the adaptive OT functionality $\mathcal{F}_{\text{AOT}}$ described in Sect. 4.4.

The constructions NOTAC and ROTAC are executed by a sender $\mathcal{E}$, an issuer $\mathcal{I}$, and receivers $\mathcal{R}$. These parties are activated through the otac.init, otac.getpolicy, otac.issue and notac.transfer or aotac.transfer interfaces. The constructions NOTAC and ROTAC execute the same program for the otac.getpolicy and otac.issue interfaces, but their programs differ in the otac.init, notac.transfer or aotac.transfer interfaces. We use the boxes $\boxed{\text{NOTAC} : \ldots}$ and $\boxed{\text{ROTAC} : \ldots}$ to describe something that is only executed in the construction NOTAC and in the construction ROTAC respectively.

1. The sender $\mathcal{E}$ is activated through the initialization interface otac.init on input the messages $(m_1, \ldots, m_N)$ and the policies $(\mathbb{P}_1, \ldots, \mathbb{P}_N)$. The sender $\mathcal{E}$ encrypts each message using the CPABE scheme. The way the message is encrypted differs in the constructions NOTAC and ROTAC.
   - In the construction NOTAC, a message m_i is encrypted using a one-time pad $B_i = H(Z_i) \oplus (0^k || m_i)$. Z_i is a random value, which is encrypted using the CPABE scheme on input the access control policy $\mathbb{P}_i$ by running $A_i \leftarrow \mathsf{ABEEnc}(par, Z_i, \mathbb{P}_i)$. A ciphertext C_i consists of (A_i, B_i).
   - In the construction ROTAC, a message m_i is first encrypted using a one-time pad $m_i' = m_i \oplus Y_i$, where Y_i is random. Then m_i' is encrypted following the same steps as in the construction NOTAC. Finally, the sender $\mathcal{E}$ creates an instance of $\mathcal{F}_{\text{AOT}}$ by invoking the aot.init interface on input $(Y_1, \ldots, Y_N)$.

   In the security proof, the use of the random oracle H allows the simulator $\mathcal{V}$ to, in the initialization phase, send the adversary $\mathcal{A}$ ciphertexts that encrypt random values and, in the transfer phase, make the adversary $\mathcal{A}$ decrypt those ciphertexts to the correct messages by programming adequately the random oracle. The simulator needs to do this when a receiver is corrupt.
   The sender also computes a zero-knowledge proof of knowledge of the master secret key *msk* of the CPABE scheme. In the security proof, this proof allows the simulator to extract *msk* and decrypt the ciphertexts sent by the adversary $\mathcal{A}$. The simulator needs to do this when the sender is corrupt.
   To end the initialization phase, the sender $\mathcal{E}$ registers with $\mathcal{F}_{\text{REG}}$ the policies, the ciphertexts, the proof and the public parameters of the scheme.
2. A receiver $\mathcal{R}$ is activated through the otac.getpolicy interface. The receiver $\mathcal{R}$ retrieves from $\mathcal{F}_{\text{REG}}$ the policies, the ciphertexts, the proof and the public parameters of the scheme. Then the receiver $\mathcal{R}$ checks the correctness of the ciphertexts by running the algorithm ABEValidCT. The receiver $\mathcal{R}$ also verifies the zero-knowledge proof of knowledge of the master secret key *msk* of the CPABE scheme. If all the checks succeed, the receiver $\mathcal{R}$ outputs the policies.
3. The issuer $\mathcal{I}$ is activated through the otac.issue interface on input a list of attributes and a receiver's identity $\mathcal{R}$. If the issuer $\mathcal{I}$ is activated for the first time, $\mathcal{I}$ creates an instance of $\mathcal{F}_{\text{AA}}$ by invoking the aa.setup interface. The issuer $\mathcal{I}$ issues the attributes to the receiver $\mathcal{R}$ by using the interface aa.issue of $\mathcal{F}_{\text{AA}}$. The universe of attributes Ψ is

$[1, L_{uni}]$, where L_{uni} is the total number of attributes. $L_{uni} + 1$ is a dummy attribute that is issued to all the receivers.

4. (i) A receiver $\mathcal{R}$ is activated through the notac.transfer interface on input a set of attributes $\mathbb{A}$. The receiver $\mathcal{R}$ aborts if those attributes were not issued by the issuer $\mathcal{I}$ to $\mathcal{R}$. Otherwise, the receiver $\mathcal{R}$ and the sender $\mathcal{E}$ run a blind key extraction protocol for the CPABE scheme. In this protocol, first the receiver $\mathcal{R}$ creates an instance of $\mathcal{F}_{\text{NOT}}$. The sender $\mathcal{E}$ computes a secret key $sk_{\mathbb{A}} = (sk_1, \ldots, sk_{L_{uni}}, sk'_{\mathbb{A}})$ for the attribute universe $[1, L_{uni}]$ by running ABEExt. The sender $\mathcal{E}$ uses the interface not.init of $\mathcal{F}_{\text{NOT}}$ on input $(sk_1, \ldots, sk_{L_{uni}}, sk'_{\mathbb{A}})$ as the messages to be transferred. Then the receiver $\mathcal{R}$ computes commitments to all the attributes in the set $\mathbb{A}$. The receiver $\mathcal{R}$ employs the aa.prove interface of $\mathcal{F}_{\text{AA}}$ in order to prove to the sender $\mathcal{E}$ that the committed attributes were issued to $\mathcal{R}$ by the issuer $\mathcal{I}$. The receiver $\mathcal{R}$ also employs the request interface not.request of $\mathcal{F}_{\text{NOT}}$ in order to obtain a secret key for the committed attributes. The sender $\mathcal{E}$ checks that the commitments received via the aa.prove.end interface of $\mathcal{F}_{\text{AA}}$ and the not.request.end interface of $\mathcal{F}_{\text{NOT}}$ are equal. If this holds, thanks to the binding property of the commitment scheme, the sender $\mathcal{E}$ is guaranteed that the attributes committed were issued by the issuer $\mathcal{I}$ and thus the receiver $\mathcal{R}$ is allowed to obtain a secret key for those attributes. After obtaining a secret for the set of attributes $\mathbb{A}$, the receiver $\mathcal{R}$ employs this secret key to decrypt each of the ciphertexts A_i. The plaintext Z_i is used to decrypt B_i. The receiver outputs all the messages for which the decryption process was successful.
(ii) A receiver $\mathcal{R}$ is activated through the aotac.transfer interface on input a set of attributes $\mathbb{A}$ and a selection value σ. If the receiver $\mathcal{R}$ still did not obtain a secret key for a set of attributes $\mathbb{A}'$ such that $\mathbb{A} \subseteq \mathbb{A}'$, then $\mathcal{R}$ executes the same program as the notac.transfer interface to obtain the messages $(m'_1, \ldots, m'_N)$. In the construction NOTAC, these are the messages that are output by the receiver, but in the construction ROTAC, they need to be further decrypted. In order to do that, the receiver $\mathcal{R}$ invokes the functionality $\mathcal{F}_{\text{AOT}}$ through the aot.transfer interface on input the selection value σ to get Y_σ. The receiver $\mathcal{R}$ outputs the message $m_\sigma = Y_\sigma \oplus m'_\sigma$.

Construction NOTAC **and Construction** ROTAC

The construction NOTAC and the construction ROTAC are parameterized with a number of messages N, a message space $\mathcal{M}$, a maximum number of attributes L_{max} and a universe of attributes Ψ. Both constructions employ a commitment scheme (CSetup, Com, VfCom), a committing CPABE with key separation scheme (ABESetup, ABEExt, ABEEnc, ABEDec, ABEValidSK, ABEValidCT), a random oracle H, a non-interactive zero-knowledge proof of knowledge scheme, and the ideal functionalities $\mathcal{F}_{\text{CRS}}^{\text{CSetup}}$, $\mathcal{F}_{\text{REG}}$, $\mathcal{F}_{\text{AA}}$ and $\mathcal{F}_{\text{NOT}}$. The construction ROTAC also employs the functionality $\mathcal{F}_{\text{AOT}}$.

1. On input (otac.init, sid, $m_1, \mathbb{P}_1, \ldots, m_N, \mathbb{P}_N$), $\mathcal{E}$ does the following.
   - Abort if $sid \neq (\mathcal{E}, \mathcal{I}, sid')$, or if $(par, msk, par_c, m_1, \mathbb{P}_1, \ldots, m_N, \mathbb{P}_N)$ is already stored, or if, for $i = 1$ to N, $m_i \notin \mathcal{M}$.
   - Send (crs.get, sid) to $\mathcal{F}_{\text{CRS}}^{\text{CSetup}}$ and wait for (crs.get.end, sid, par_c) from $\mathcal{F}_{\text{CRS}}^{\text{CSetup}}$.
   - Run $(par, msk) \leftarrow \text{ABESetup}(1^k)$.
   - Compute a proof $\pi = \text{PK}\{(msk) : (par, msk) \leftarrow \text{ABESetup}(1^k)\}$.
   - For $i = 1$ to N, do the following:
     - NOTAC : Set $m'_i = m_i$.
     - ROTAC : Pick random Y_i and set $m'_i = m_i \oplus Y_i$.
     - Pick random Z_i and run $A_i \leftarrow \text{ABEEnc}(par, Z_i, \mathbb{P}_i)$.

- Set $B_i = H(Z_i) \oplus (0^k || m'_i)$.
- Set $C_i = (A_i, B_i)$.

- ROTAC: Parse sid as $(\mathcal{E}, \mathcal{I}, sid')$, set $sid_{\text{AOT}} = (\mathcal{E}, sid')$, send (aot.init, sid_{AOT}, $Y_1, \ldots, Y_N$) to $\mathcal{F}_{\text{AOT}}$ and wait for (aot.init.end, sid_{AOT}) from $\mathcal{F}_{\text{AOT}}$.

- Store $(par, msk, par_c, m_1, \mathbb{P}_1, \ldots, m_N, \mathbb{P}_N)$.
- Send (reg.register, sid, $\langle par, \pi, C_1, \mathbb{P}_1 \ldots, C_N, \mathbb{P}_N \rangle$) to $\mathcal{F}_{\text{REG}}$ and receive (reg.register.end, sid) from $\mathcal{F}_{\text{REG}}$.
- Output (otac.init.end, sid).

2. On input (otac.getpolicy, sid), $\mathcal{R}$ does the following:
   - Abort if $(par_c, par, C_1, \mathbb{P}_1 \ldots, C_N, \mathbb{P}_N)$ is not stored.
   - Send (crs.get, sid) to $\mathcal{F}_{\text{CRS}}^{\text{CSetup}}$ and wait for (crs.get.end, sid, par_c) from $\mathcal{F}_{\text{CRS}}^{\text{CSetup}}$.
   - Send (reg.retrieve, sid) to $\mathcal{F}_{\text{REG}}$ and wait for (reg.retrieve.end, sid, $\langle par, \pi, C_1, \mathbb{P}_1 \ldots, C_N, \mathbb{P}_N \rangle$) from $\mathcal{F}_{\text{REG}}$.
   - Abort if the proof π is not correct.
   - For $i = 1$ to N, do the following:
     - Parse C_i as (A_i, B_i).
     - Abort if $1 \neq \mathsf{ABEValidCT}(par, A_i)$.
   - Store $(par_c, par, C_1, \mathbb{P}_1 \ldots, C_N, \mathbb{P}_N)$.
   - Output (otac.getpolicy.end, sid, $\mathbb{P}_1, \ldots, \mathbb{P}_N$).
3. On input (otac.issue, sid, $\mathcal{R}$, $\langle a_l \rangle_{l=1}^{L}$), $\mathcal{I}$ and $\mathcal{R}$ do the following:
   - Abort if $L > L_{max}$ or if $\langle a_l \rangle_{l=1}^{L} \nsubseteq [1, L_{uni}]$.
   - If par_c is not stored, $\mathcal{I}$ does the following:
     - Send (crs.get, sid) to $\mathcal{F}_{\text{CRS}}^{\text{CSetup}}$, wait for (crs.get.end, sid, par_c) from $\mathcal{F}_{\text{CRS}}^{\text{CSetup}}$ and store par_c.
     - Parse sid as $(\mathcal{E}, \mathcal{I}, sid')$, set $sid_{\text{AA}} = (\mathcal{I}, sid')$ and send (aa.setup, sid_{AA}, par_c, VfCom) to $\mathcal{F}_{\text{AA}}$ and wait for a message (aa.setup.end, sid) from $\mathcal{F}_{\text{AA}}$.
   - $\mathcal{I}$ parses sid as $(\mathcal{E}, \mathcal{I}, sid')$, sets $sid_{\text{AA}} = (\mathcal{I}, sid')$ and sends (aa.issue, sid_{AA}, $\mathcal{R}$, $[\langle a_l \rangle_{l=1}^{L}, L_{uni} + 1]$) to $\mathcal{F}_{\text{AA}}$.
   - $\mathcal{R}$ receives (aa.issue.end, sid_{AA}, $[\langle a_l \rangle_{l=1}^{L}, L_{uni} + 1]$) from $\mathcal{F}_{\text{AA}}$.
   - $\mathcal{R}$ stores $[\langle a_l \rangle_{l=1}^{L}, L_{uni} + 1]$.
   - $\mathcal{R}_j$ outputs (otac.issue.end, sid, $\langle a_l \rangle_{l=1}^{L}$).
4. On input (notac.transfer, sid, $\mathbb{A}$), $\mathcal{R}$ and $\mathcal{E}$ do the following:
   - $\mathcal{R}$ aborts if there is no attribute set $[\langle a_l \rangle_{l=1}^{L}, L_{uni} + 1]$ stored such that $\mathbb{A} \subseteq \langle a_l \rangle_{l=1}^{L}$.
   - $\mathcal{R}$ aborts if $(par_c, par, C_1, \mathbb{P}_1 \ldots, C_N, \mathbb{P}_N)$ is not stored.
   - If there is a tuple $(\mathbb{A}', m'_1, \ldots, m'_N)$ stored such that $\mathbb{A} \subseteq \mathbb{A}'$, $\mathcal{R}$ outputs (notac.transfer.end, sid, $m'_1, \ldots, m'_N$), else executes the remaining steps.
   - $\mathcal{R}$ sets $sid_{\text{NOT}} = (\mathcal{E}, sid'')$ for a fresh sid'' and sends (not.initrec, sid_{NOT}) to $\mathcal{F}_{\text{NOT}}$.
   - $\mathcal{E}$ receives (not.initrec.end, sid_{NOT}) from $\mathcal{F}_{\text{NOT}}$.
   - $\mathcal{E}$ sets $\Psi = [1, L_{uni}]$ and runs $sk_{\mathbb{A}} \leftarrow \mathsf{ABEExt}(msk, \Psi)$, where $sk_{\mathbb{A}} = (sk'_{\mathbb{A}}, sk_1, \ldots, sk_{L_{uni}})$.
   - $\mathcal{E}$ sends (not.init, sid_{NOT}, $\langle sk_1, \ldots, sk_{L_{uni}}, sk'_{\mathbb{A}} \rangle$, par_c, VfCom) to $\mathcal{F}_{\text{NOT}}$.
   - $\mathcal{R}$ receives (not.init.end, sid_{NOT}, par_c, VfCom) from $\mathcal{F}_{\text{NOT}}$.
   - Set $\mathbb{A} = \mathbb{A} \cup \{L_{uni} + 1\}$.
   - For all $a_l \in \mathbb{A}$, $\mathcal{R}$ runs $(com_l, open_l) \leftarrow \mathsf{Com}(par_c, a_l)$.
   - $\mathcal{R}$ sends (not.request, sid_{NOT}, $\langle com_l, a_l, open_l \rangle_{\forall\ l\ st\ a_l \in \mathbb{A}}$) to $\mathcal{F}_{\text{NOT}}$.
   - $\mathcal{R}$ parses sid as $(\mathcal{E}, \mathcal{I}, sid')$, sets $sid_{\text{AA}} = (\mathcal{I}, sid')$ and sends (aa.prove, sid_{AA}, $\mathcal{E}$, $\langle com_l, a_l, open_l \rangle_{\forall\ l\ st\ a_l \in \mathbb{A}}$) to $\mathcal{F}_{\text{AA}}$.

- $\mathcal{E}$ receives (not.request.end, sid_{NOT}, $\langle com_l \rangle_{\forall\ l\ st\ a_l \in \mathbb{A}}$) from $\mathcal{F}_{\text{NOT}}$.
- $\mathcal{E}$ receives (aa.prove.end, sid_{AA}, $\langle com'_l \rangle_{\forall\ l\ st\ a_l \in \mathbb{A}}$) from $\mathcal{F}_{\text{AA}}$.
- $\mathcal{E}$ aborts if $\langle com_l \rangle_{\forall\ l\ st\ a_l \in \mathbb{A}} \neq \langle com'_l \rangle_{\forall\ l\ st\ a_l \in \mathbb{A}}$.
- $\mathcal{E}$ sends (not.transfer, sid_{NOT}, $\langle com_l \rangle_{\forall\ l\ st\ a_l \in \mathbb{A}}$) to $\mathcal{F}_{\text{NOT}}$.
- $\mathcal{R}$ receives (not.transfer.end, sid, $\langle a_l, sk_{a_l} \rangle_{\forall\ a_l \in \mathbb{A}}$).
- $\mathcal{R}$ sets $sk_{\mathbb{A}} = \langle sk_{a_l} \rangle_{\forall\ a_l \in \mathbb{A}}$.
- $\mathcal{R}$ aborts if $1 \neq \mathsf{ABEValidSK}(par, \mathbb{A}, sk_{\mathbb{A}})$.
- For $i = 1$ to N, $\mathcal{R}$ does the following:
  - Parse C_i as (A_i, B_i).
  - Run $Z_i \leftarrow \mathsf{ABEDec}(par, A_i, sk_{\mathbb{A}})$.
  - Set $D_i = H(Z_i) \oplus B_i$.
  - Parse D_i as $(u||x)$. If $u = 0^k$, set $m'_i = x$, else set $m'_i = \perp$.
- $\mathcal{R}$ outputs (notac.transfer.end, sid, $m'_1, \ldots, m'_N$).

ROTAC :

On input (aotac.transfer, sid, $\mathbb{A}$, σ), $\mathcal{R}$ and $\mathcal{E}$ do the following.

- If there is not a tuple $(\mathbb{A}', m'_1, \ldots, m'_N)$ stored such that $\mathbb{A} \subseteq \mathbb{A}'$, the receiver $\mathcal{R}$ runs (notac.transfer, sid, $\mathbb{A}$) and obtains (notac.transfer.end, sid, $m'_1, \ldots, m'_N$).
- $\mathcal{R}$ aborts if $m'_\sigma = \perp$.
- $\mathcal{R}$ parses sid as $(\mathcal{E}, \mathcal{I}, sid')$, sets $sid_{\text{AOT}} = (\mathcal{E}, sid')$, sends the message (aot.transfer, sid_{AOT}, σ) to $\mathcal{F}_{\text{AOT}}$ and waits for the message (aot.transfer.end, sid_{AOT}, Y_σ) from $\mathcal{F}_{\text{AOT}}$.
- $\mathcal{R}$ computes $m_\sigma = m'_\sigma \oplus Y_\sigma$.
- $\mathcal{R}$ outputs (aotac.transfer.end, sid, m_σ).

5.3 Security analysis of oblivious transfer with access control

Theorem 1 *The construction* NOTAC *and the construction* ROTAC *securely realize* $\mathcal{F}_{\text{NOTAC}}$ *and* $\mathcal{F}_{\text{ROTAC}}$ *respectively in the* $(\mathcal{F}_{\text{CRS}}^{\mathsf{CSetup}}, \mathcal{F}_{\text{REG}}, \mathcal{F}_{\text{AA}}, \mathcal{F}_{\text{NOT}})$*-hybrid model and in the random oracle model if the committing* CPABE *with key separation scheme is secure as defined in Definition 7, if the commitment scheme is hiding and binding, and if the zero-knowledge proof of knowledge scheme fulfills the properties of zero-knowledge and extractability defined in Sect. 2.3. The construction* ROTAC *also operates in the* $\mathcal{F}_{\text{AOT}}$*-hybrid model.*

To prove that the construction NOTAC and the construction ROTAC securely realize the ideal functionalities $\mathcal{F}_{\text{NOTAC}}$ and $\mathcal{F}_{\text{ROTAC}}$ respectively, we have to show that for any environment $\mathcal{Z}$ and any adversary $\mathcal{A}$ there exists a simulator $\mathcal{V}$, such that $\mathcal{Z}$ cannot distinguish whether it is interacting with $\mathcal{A}$ and the protocol in the real world or with $\mathcal{V}$ and $\mathcal{F}_{\text{NOTAC}}$ (respectively, $\mathcal{F}_{\text{ROTAC}}$). The simulator thereby plays the role of all honest parties in the real world and interacts with $\mathcal{F}_{\text{NOTAC}}$ (respectively, $\mathcal{F}_{\text{ROTAC}}$) for all corrupt parties in the ideal world.

Our simulator $\mathcal{V}$ employs any simulator $\mathcal{V}_{\text{CRS}}$, $\mathcal{V}_{\text{REG}}$, $\mathcal{V}_{\text{AA}}$ and $\mathcal{V}_{\text{NOT}}$ for the constructions that realize $\mathcal{F}_{\text{CRS}}^{\mathsf{CSetup}}$, $\mathcal{F}_{\text{REG}}$, $\mathcal{F}_{\text{AA}}$ and $\mathcal{F}_{\text{NOT}}$ respectively. We note that the simulators for all the constructions that realize $\mathcal{F}_{\text{CRS}}^{\mathsf{CSetup}}$, $\mathcal{F}_{\text{REG}}$, $\mathcal{F}_{\text{AA}}$ and $\mathcal{F}_{\text{NOT}}$ communicate with each of those functionalities through the same interfaces. These are the interfaces that our simulator employs to communicate with $\mathcal{V}_{\text{CRS}}$, $\mathcal{V}_{\text{REG}}$, $\mathcal{V}_{\text{AA}}$ and $\mathcal{V}_{\text{NOT}}$. $\mathcal{V}$ forwards all the messages exchanged between $\mathcal{V}_{\text{CRS}}$, $\mathcal{V}_{\text{REG}}$, $\mathcal{V}_{\text{AA}}$ and $\mathcal{V}_{\text{NOT}}$ and the adversary $\mathcal{A}$. When $\mathcal{A}$ sends a

message that corresponds to a protocol that realizes $\mathcal{F}_{\text{CRS}}^{\text{CSetup}}$, $\mathcal{F}_{\text{REG}}$, $\mathcal{F}_{\text{AA}}$ or $\mathcal{F}_{\text{NOT}}$, $\mathcal{V}$ implicitly forwards that message to the respective simulator $\mathcal{V}_{\text{CRS}}$, $\mathcal{V}_{\text{REG}}$, $\mathcal{V}_{\text{AA}}$ or $\mathcal{V}_{\text{NOT}}$. In the construction ROTAC, $\mathcal{V}$ also employs any simulator $\mathcal{V}_{\text{AOT}}$ that realizes $\mathcal{F}_{\text{AOT}}$.

Case $\mathcal{E}$ corrupt. We first describe the simulator $\mathcal{V}$ for the case in which the sender is corrupt. In our simulation, we employ the extractor algorithm $\mathcal{E}$ for proof of knowledge π of the non-interactive zero-knowledge proof of knowledge scheme described in Sect. 2.3. We use the boxes $\boxed{\text{NOTAC} : \ldots}$ and $\boxed{\text{ROTAC} : \ldots}$ to describe something that is only executed in the simulations of the constructions NOTAC and ROTAC respectively.

- On input (crs.get, *sid*) from $\mathcal{V}_{\text{CRS}}$, if par_c is not stored, the simulator $\mathcal{V}$ runs $par_c \leftarrow \mathsf{CSetup}(1^k)$. The simulator $\mathcal{V}$ sends (crs.get, *sid*, par_c) to $\mathcal{V}_{\text{CRS}}$. After receiving (crs.get, *sid*) from $\mathcal{V}_{\text{CRS}}$, $\mathcal{V}$ sends (crs.get.end, *sid*, par_c) to $\mathcal{V}_{\text{CRS}}$.
- On input a random oracle query q_i from $\mathcal{A}$, $\mathcal{V}$ runs the zero knowledge simulator $(h_i, st) \leftarrow \mathcal{S}(1, st, q_i)$ and sends h_i to $\mathcal{A}$. The state st is initialized to $\emptyset$ before the first time the zero-knowledge simulator is run. The pair (q_i, h_i) is stored in $\mathcal{T}_H$.
- On input a random oracle query Z_i from $\mathcal{A}$, $\mathcal{V}$ replies with a consistent random value $H(Z_i)$ and records $[Z_i, H(Z_i)]$.

- ROTAC: On input (aot.init, sid_{AOT}, $Y_1, \ldots, Y_N$) from $\mathcal{V}_{\text{AOT}}$, store $(Y_1, \ldots, Y_N)$ and send (aot.init, sid_{AOT}) to $\mathcal{V}_{\text{AOT}}$.

- On input (reg.register, *sid*, $\langle par, \pi, C_1, \mathbb{P}_1 \ldots, C_N, \mathbb{P}_N \rangle$) from $\mathcal{V}_{\text{REG}}$, the simulator $\mathcal{V}$ does the following:
  - Run the extractor $\mathcal{E}_{\mathcal{A}}(\langle \Gamma, H \rangle, \pi; \rho, \mathcal{T}_H, \mathcal{T})$ in order to extract the master secret key *msk* of the CPABE scheme.
  - Set $\Psi = [1, L_{uni}]$ and run $sk_{\mathbb{A}} \leftarrow \mathsf{ABEExt}(msk, \Psi)$ to get a secret key $sk_{\mathbb{A}}$ able to decrypt every ciphertext.
  - For $i = 1$ to N, parse C_i as (A_i, B_i), run $b \leftarrow \mathsf{ABEValidCT}(par, A_i)$ and, if $b = 0$, set $m_i = \perp$.
  - For $i = 1$ to N, run $Z_i \leftarrow \mathsf{ABEDec}(par, A_i, sk_{\mathbb{A}})$ and decrypt the message $u||x = B_i \oplus H(Z_i)$. If $u = 0^k$, set $m_i = x$, else set $m_i = \perp$.
  - $\boxed{\text{ROTAC : Compute } m_i = m_i \oplus Y_i \text{ for } i = 1 \text{ to } N.}$
  - Send (otac.init, *sid*, $m_1, \mathbb{P}_1, \ldots, m_N, \mathbb{P}_N$) to the functionality $\mathcal{F}_{\text{NOTAC}}$ or the functionality $\mathcal{F}_{\text{ROTAC}}$ and receive (otac.init, *sid*, $\mathbb{P}_1, \ldots, \mathbb{P}_N$) from $\mathcal{F}_{\text{NOTAC}}$ or $\mathcal{F}_{\text{ROTAC}}$.
  - Send (reg.register, *sid*, $\langle par, \pi, C_1, \mathbb{P}_1 \ldots, C_N, \mathbb{P}_N \rangle$) to $\mathcal{V}_{\text{REG}}$ and wait for the message (reg.register, *sid*) from $\mathcal{V}_{\text{REG}}$.
  - Send the message (otac.init, *sid*) to the functionality $\mathcal{F}_{\text{NOTAC}}$ or the functionality $\mathcal{F}_{\text{ROTAC}}$ and receive (otac.init.end, *sid*) from $\mathcal{F}_{\text{NOTAC}}$ or $\mathcal{F}_{\text{ROTAC}}$.
  - Send (reg.register.end, *sid*) to $\mathcal{V}_{\text{REG}}$.
- On input (otac.getpolicy, *sid*, $\mathbb{P}_1, \ldots, \mathbb{P}_N$) from $\mathcal{F}_{\text{NOTAC}}$ or $\mathcal{F}_{\text{ROTAC}}$, the simulator $\mathcal{V}$ sends (crs.get, *sid*, par_c) to $\mathcal{V}_{\text{CRS}}$. Upon receiving (crs.get, *sid*) from $\mathcal{V}_{\text{CRS}}$, the simulator $\mathcal{V}$ sends the message (reg.retrieve, *sid*, $\langle par, \pi, C_1, \mathbb{P}_1 \ldots, C_N, \mathbb{P}_N \rangle$) to $\mathcal{V}_{\text{REG}}$. Upon receiving (reg.retrieve, *sid*) from $\mathcal{V}_{\text{REG}}$, the simulator $\mathcal{V}$ sends (otac.getpolicy, *sid*) to $\mathcal{F}_{\text{NOTAC}}$ or $\mathcal{F}_{\text{ROTAC}}$.
- On input (otac.issue, *sid*, $\mathcal{R}$) from $\mathcal{F}_{\text{NOTAC}}$ or $\mathcal{F}_{\text{ROTAC}}$, if this is the first message (otac.issue, *sid*, $\cdot$) received, the simulator $\mathcal{V}$ does the following:

- Send (crs.get, sid, par_c) to $\mathcal{V}_{\text{CRS}}$ and wait for (crs.get, sid) from $\mathcal{V}_{\text{CRS}}$.
- Parse sid as $(\mathcal{E}, \mathcal{I}, sid')$, set $sid_{\text{AA}} = (\mathcal{I}, sid')$ and send (aa.setup, sid_{AA}, par_c, VfCom) to $\mathcal{V}_{\text{AA}}$ and wait for (aa.setup, sid) from $\mathcal{V}_{\text{AA}}$.

The simulator $\mathcal{V}$ sends (aa.issue, sid, $\mathcal{R}$) to $\mathcal{V}_{\text{AA}}$ and waits for (aa.issue, sid) from $\mathcal{V}_{\text{AA}}$. $\mathcal{V}$ sends (otac.issue, sid) to $\mathcal{F}_{\text{NOTAC}}$ or $\mathcal{F}_{\text{ROTAC}}$.

- On input (notac.transfer, sid, $|\mathbb{A}|$) from $\mathcal{F}_{\text{NOTAC}}$, the simulator $\mathcal{V}$ does the following:
  - Set $sid_{\text{NOT}} = (\mathcal{E}, sid'')$ for a fresh sid'', send (not.initrec.end, sid_{NOT}) to $\mathcal{V}_{\text{NOT}}$ and wait for (not.init, sid_{NOT}, $\langle sk_1, \ldots, sk_{L_{uni}}, sk'_{\mathbb{A}} \rangle$, par_c, VfCom) from $\mathcal{V}_{\text{NOT}}$.
  - Set $sk_{\mathbb{A}} = \langle sk_1, \ldots, sk_{L_{uni}}, sk'_{\mathbb{A}} \rangle$ and, if $1 \neq \mathsf{ABEValidSK}(par, \mathbb{A}, sk_{\mathbb{A}})$, send (notac.transfer, sid, 0) to $\mathcal{F}_{\text{NOTAC}}$.
  - Else, $\mathcal{V}$ employs $sk_{\mathbb{A}}$ to decrypt the ciphertexts $(C_1, \mathbb{P}_1, \ldots, C_N, \mathbb{P}_N)$ and checks if the resulting messages equal those sent to $\mathcal{F}_{\text{NOTAC}}$. If not, $\mathcal{V}$ aborts.
  - Else, $\mathcal{V}$ computes commitments $\langle com_l \rangle_{l=1}^{|\mathbb{A}|+1}$ to random values and sends the message (not.request.end, sid_{NOT}, $\langle com_l \rangle_{l=1}^{|\mathbb{A}|+1}$) to $\mathcal{V}_{\text{NOT}}$ and (aa.prove.end, sid_{AA}, $\langle com_l \rangle_{l=1}^{|\mathbb{A}|+1}$) to $\mathcal{V}_{\text{AA}}$.
  - Upon receiving (not.transfer, sid_{NOT}, $\langle com_l \rangle_{l=1}^{|\mathbb{A}|+1}$) from $\mathcal{V}_{\text{NOT}}$, if $\langle com_l \rangle_{l=1}^{|\mathbb{A}|+1}$ are equal to the commitments sent to $\mathcal{V}_{\text{NOT}}$, $\mathcal{V}$ sends (notac.transfer, sid, 1) to the functionality $\mathcal{F}_{\text{NOTAC}}$, else sends (notac.transfer, sid, 0) to $\mathcal{F}_{\text{NOTAC}}$.

- ROTAC: On input (aotac.transfer, sid, $|\mathbb{A}|$) from $\mathcal{F}_{\text{ROTAC}}$, if $|\mathbb{A}| \neq 0$, $\mathcal{V}$ runs its code on input (notac.transfer, sid, $|\mathbb{A}|$), gets (notac.transfer, sid, b), and if $b = 0$, $\mathcal{V}$ sends (notac.transfer, sid, 0) to $\mathcal{F}_{\text{ROTAC}}$. Else, $\mathcal{V}$ parses sid as $(\mathcal{E}, \mathcal{I}, sid')$, sets $sid_{\text{AOT}} = (\mathcal{E}, sid')$ and sends (aot.transfer, sid) to $\mathcal{V}_{\text{AOT}}$. After receiving (aot.transfer, sid, b) from $\mathcal{V}_{\text{AOT}}$, $\mathcal{V}$ sends (aotac.transfer, sid, b) to $\mathcal{F}_{\text{ROTAC}}$.

Theorem 2 *When the sender $\mathcal{E}$ is corrupt, the construction* NOTAC *and the construction* ROTAC *securely realize the functionalities $\mathcal{F}_{\text{NOTAC}}$ and $\mathcal{F}_{\text{ROTAC}}$ respectively in the $(\mathcal{F}_{\text{CRS}}^{\text{CSetup}}, \mathcal{F}_{\text{REG}}, \mathcal{F}_{\text{AA}}, \mathcal{F}_{\text{NOT}})$-hybrid model and in the random oracle model if the zero-knowledge proof of knowledge scheme fulfills the extractability property defined in Sect. 2.3, if the* CPABE *scheme is committing, and if the commitment scheme fulfills the hiding property defined in Sect. 2.4. The construction* ROTAC *also operates in the $\mathcal{F}_{\text{AOT}}$-hybrid model.*

Proof We show by means of a series of hybrid games that the environment $\mathcal{Z}$ cannot distinguish between the real and the ideal ensemble with non-negligible probability. We denote by Pr [**Game** i] the probability that the environment distinguishes **Game** i from the real world protocol.

Game 0 This game corresponds to the execution of the real world protocol. Therefore, we have that Pr [**Game** 0] = 0.

Game 1 **Game** 1 follows **Game** 0, except that **Game** 1 replies the random oracle queries as described in our simulation and extracts the witness msk from the zero-knowledge proof of knowledge π. The extraction property implied by the simulation extractability property ensures that extraction works with extraction error ν. Therefore, $|\Pr[\textbf{Game } 1] - \Pr[\textbf{Game } 0]| \leq \text{Adv}_{\mathcal{A}}^{\text{we-zkpk}}$.

Game 2 **Game** 2 follows **Game** 1, except that **Game** 2 aborts if the result of decrypting a ciphertext with a secret key for all the attributes computed on input msk is

different from the result of decrypting a ciphertext with the secret key input by the adversary to $\mathcal{F}_{\text{NOT}}$. If the CPABE scheme is committing, **Game** 2 aborts with negligible probability. Therefore, $|\Pr[\textbf{Game } 2] - \Pr[\textbf{Game } 1]| \leq \text{Adv}_{\mathcal{A}}^{\text{com-cpabe}}$.

Game 3 **Game** 3 follows **Game** 2, except that **Game** 3 replaces the commitments sent to the adversary via $\mathcal{F}_{\text{NOT}}$ and $\mathcal{F}_{\text{AA}}$ by commitments to random messages. The hiding property of the commitment scheme implies that the adversary distinguishes commitments to known messages from commitments to random messages with negligible probability. Therefore, $|\Pr[\textbf{Game } 3] - \Pr[\textbf{Game } 2]| \leq \text{Adv}_{\mathcal{A}}^{\text{hid-com}}$.

The distribution of **Game** 3 is identical to our simulation. □

Case $\mathcal{R}$ corrupt. We now describe the simulator $\mathcal{V}$ for the case in which (a subset of) the receivers are corrupt. In our simulation, we employ the simulation algorithm $\mathcal{S}$ of the non-interactive zero-knowledge proof of knowledge scheme described in Sect. 2.3.

- On input a random oracle query q_i from $\mathcal{A}$, $\mathcal{V}$ runs the zero knowledge simulator $(h_i, st) \leftarrow \mathcal{S}(1, st, q_i)$ and sends h_i to $\mathcal{A}$. The state st is initialized to $\emptyset$ before the first time the zero-knowledge simulator is run. The pair (q_i, h_i) is stored in $\mathcal{T}_H$.
- On input (otac.init, $sid, \mathbb{P}_1, \ldots, \mathbb{P}_N$) from $\mathcal{F}_{\text{NOTAC}}$ or $\mathcal{F}_{\text{ROTAC}}$, $\mathcal{V}$ does the following:
  - Run $(par, msk) \leftarrow \mathsf{ABESetup}(1^k)$.
  - Run the zero-knowledge simulator $(\pi, st) \leftarrow \mathcal{S}(2, st, [par])$ to obtain a simulated proof π for the language $\mathsf{PK}\{(msk) : (par, msk) \leftarrow \mathsf{ABESetup}(1^k)\}$.
  - For $i = 1$ to N, pick random (Z_i, B_i), encrypt $A_i \leftarrow \mathsf{ABEEnc}(par, Z_i, \mathbb{P}_i)$ and set $C_i = (A_i, B_i)$.

  - ROTAC: Parse sid as $(\mathcal{E}, \mathcal{I}, sid')$, sets sid_{AOT} to $(\mathcal{E}, sid')$, sends (aot.init, sid_{AOT}) to $\mathcal{V}_{\text{AOT}}$ and waits for (aot.init, sid_{AOT}) from $\mathcal{V}_{\text{AOT}}$.

  - Send the message (reg.register, $sid, \langle par, \pi, C_1, \mathbb{P}_1, \ldots, C_N, \mathbb{P}_N\rangle$) to $\mathcal{V}_{\text{REG}}$ and wait for the message (reg.register, sid) from $\mathcal{V}_{\text{REG}}$.
  - Store $(msk, par, \pi, Z_1, C_1, \mathbb{P}_1 \ldots, Z_N, C_N, \mathbb{P}_N)$.
  - Send (otac.init, sid) to $\mathcal{F}_{\text{NOTAC}}$ or $\mathcal{F}_{\text{ROTAC}}$.
- On input (crs.get, sid) from $\mathcal{V}_{\text{CRS}}$, if par_c is not stored, the simulator $\mathcal{V}$ runs $par_c \leftarrow \mathsf{CSetup}(1^k)$. The simulator $\mathcal{V}$ sends (crs.get, sid, par_c) to $\mathcal{V}_{\text{CRS}}$. After receiving (crs.get, sid) from $\mathcal{V}_{\text{CRS}}$, the simulator $\mathcal{V}$ sends (crs.get, sid, par_c) to $\mathcal{V}_{\text{CRS}}$.
- On input (reg.retrieve, sid) from $\mathcal{V}_{\text{REG}}$, if $sid = (\mathcal{E}, \mathcal{R}, sid')$ and $(msk, par, \pi, Z_1, C_1, \mathbb{P}_1 \ldots, Z_N, C_N, \mathbb{P}_N)$ is already stored, $\mathcal{V}$ sends (reg.retrieve, $sid, \langle par, \pi, C_1, \mathbb{P}_1, \ldots, C_N, \mathbb{P}_N\rangle$) to $\mathcal{V}_{\text{REG}}$ and waits for (reg.retrieve, sid) from $\mathcal{V}_{\text{REG}}$. $\mathcal{V}$ sends (reg.retrieve.end, $sid, \langle par, \pi, C_1, \mathbb{P}_1, \ldots, C_N, \mathbb{P}_N\rangle$) to $\mathcal{V}_{\text{REG}}$.
- On input (otac.issue, $sid, \mathcal{R}$) from $\mathcal{F}_{\text{NOTAC}}$ or $\mathcal{F}_{\text{ROTAC}}$, if this is the first message (otac.issue, $sid, \cdot$) received, the simulator $\mathcal{V}$ does the following:
  - Send (crs.get, sid, par_c) to $\mathcal{V}_{\text{CRS}}$ and wait for (crs.get, sid) from $\mathcal{V}_{\text{CRS}}$.
  - Parse sid as $(\mathcal{E}, \mathcal{I}, sid')$, set $sid_{\text{AA}} = (\mathcal{I}, sid')$, send (aa.setup, sid_{AA}, par_c, VfCom) to $\mathcal{V}_{\text{AA}}$ and wait for (aa.setup, sid) from $\mathcal{V}_{\text{AA}}$.

  The simulator $\mathcal{V}$ sends (aa.issue, $sid, \mathcal{R}$) to $\mathcal{V}$ and waits for (aa.issue, sid) from $\mathcal{V}$. $\mathcal{V}$ sends (otac.issue, sid) to $\mathcal{F}_{\text{NOTAC}}$ or $\mathcal{F}_{\text{ROTAC}}$.
- Upon receiving (otac.issue.end, $sid, \langle a_l\rangle_{l=1}^{L}$) from $\mathcal{F}_{\text{NOTAC}}$ or $\mathcal{F}_{\text{ROTAC}}$, $\mathcal{V}$ sends (aa.issue.end, $sid_{\text{AA}}, [\langle a_l\rangle_{l=1}^{L}, L_{uni} + 1]$) to $\mathcal{V}_{\text{AA}}$.

– On input (notac.transfer, sid, $|\mathbb{A}|$) from the functionality $\mathcal{F}_{\text{NOTAC}}$, $\mathcal{V}$ sets $sid_{\text{NOT}} = (\mathcal{E}, sid'')$ for a fresh sid'' and sends the message (not.initrec, sid_{NOT}) to $\mathcal{V}_{\text{NOT}}$. Upon receiving (not.initrec, sid_{NOT}) from $\mathcal{V}_{\text{NOT}}$, $\mathcal{V}$ sends the message (not.init, sid_{NOT}, par_c, VfCom) to $\mathcal{V}_{\text{NOT}}$ and waits for the message (not.init, sid_{NOT}) from $\mathcal{V}_{\text{NOT}}$. $\mathcal{V}$ sends the message (not.request, sid) to $\mathcal{V}_{\text{NOT}}$ and waits for the message (not.request, sid) from $\mathcal{V}_{\text{NOT}}$. $\mathcal{V}$ sends the message (aa.prove.end, sid, $\mathcal{E}$) to $\mathcal{V}_{\text{AA}}$ and waits for the message (aa.prove.end, sid) from $\mathcal{V}_{\text{AA}}$. $\mathcal{V}$ sends the message (not.transfer, sid) to $\mathcal{V}_{\text{NOT}}$ and waits for the message (not.transfer, sid) from $\mathcal{V}_{\text{NOT}}$. $\mathcal{V}$ sends the message (notac.transfer, sid) to $\mathcal{F}_{\text{NOTAC}}$.
– On input the message (not.initrec, sid_{NOT}) from $\mathcal{V}_{\text{NOT}}$, if $\langle msk, par, \pi, C_1, \mathbb{P}_1 \ldots, C_N, \mathbb{P}_N \rangle$ is stored, $\mathcal{V}$ sends the message (not.init.end, sid_{NOT}, par_c, VfCom) to $\mathcal{V}_{\text{NOT}}$.
– On input the message (not.request, sid_{NOT}, $\langle com_l, a_l, open_l \rangle_{\forall\ l\ st\ a_l \in \mathbb{A}}$) and the message (aa.prove, sid_{AA}, $\mathcal{E}$, $\langle com'_l, a'_l, open_l \rangle_{\forall\ l\ st\ a_l \in \mathbb{A}}$) from $\mathcal{V}_{\text{AA}}$, the simulator $\mathcal{V}$ does nothing if the commitments $\langle com_l \rangle_{\forall\ l\ st\ a_l \in \mathbb{A}}$ and the commitments $\langle com'_l \rangle_{\forall\ l\ st\ a_l \in \mathbb{A}}$ are not equal. Else, the simulator $\mathcal{V}$ aborts if the commitments are equal but $\langle a'_l \rangle_{\forall\ l\ st\ a_l \in \mathbb{A}}$ and $\langle a'_l \rangle_{\forall\ l\ st\ a_l \in \mathbb{A}}$ are not equal. Otherwise the simulator $\mathcal{V}$ sends the message (notac.transfer, sid, $\mathbb{A}$) to the functionality $\mathcal{F}_{\text{NOTAC}}$ and receives the message (notac.transfer.end, sid, $\langle m_k \rangle_{k=1}^{N}$) from the functionality $\mathcal{F}_{\text{NOTAC}}$. $\mathcal{V}$ stores $\langle m_k \rangle_{k=1}^{N}$. $\mathcal{V}$ runs $(sk'_{\mathbb{A}}, \text{st}) \leftarrow \text{ABEExtInit}(msk)$ and, for each attribute $a_l \in \mathbb{A}$, $\mathcal{V}$ runs $sk_a \leftarrow \text{ABEExtAtt}(\text{st}, a)$. $\mathcal{V}$ sets $sk_{L_{uni}+1} = sk'_{\mathbb{A}}$ and $\mathbb{A} \leftarrow \mathbb{A} \cup \{L_{uni} + 1\}$. $\mathcal{V}$ sends the message (not.transfer.end, sid_{NOT}, $\langle a_l, sk_{a_l} \rangle_{\forall\ l\ st\ a_l \in \mathbb{A}}$) to $\mathcal{V}_{\text{NOT}}$.

– NOTAC : On input a random oracle query Z_i from $\mathcal{A}$, if $\mathcal{V}$ does not store Z_i in the tuple $(msk, par, \pi, Z_1, C_1, \mathbb{P}_1, \ldots, Z_N, C_N, \mathbb{P}_N)$, $\mathcal{V}$ replies with a consistent random value. Else, if $m_i = \perp$, $\mathcal{V}$ aborts. Else, $\mathcal{V}$ sends $H(Z_i) = B_i \oplus (0^k || m_i)$.

– ROTAC : On input (aotac.transfer, sid, $|\mathbb{A}|$) from $\mathcal{F}_{\text{ROTAC}}$, if $|\mathbb{A}| \neq 0$, the simulator $\mathcal{V}$ executes (notac.transfer, sid, $|\mathbb{A}|$). Then $\mathcal{V}$ sends (aot.transfer, sid) to $\mathcal{V}_{\text{AOT}}$ and waits for the message (aot.transfer, sid) from $\mathcal{V}_{\text{AOT}}$. $\mathcal{V}$ sends (aotac.transfer, sid) to $\mathcal{F}_{\text{ROTAC}}$.
– ROTAC : On input a random oracle query Z_i from $\mathcal{A}$, if $\mathcal{V}$ does not store Z_i in the tuple $(msk, par, \pi, Z_1, C_1, \mathbb{P}_1, \ldots, Z_N, C_N, \mathbb{P}_N)$, $\mathcal{V}$ replies with a consistent random value. Else, if $m_i = \perp$, $\mathcal{V}$ aborts. Else, $\mathcal{V}$ picks random Y_i and sends $H(Z_i) = B_i \oplus (0^k || (m_i \oplus Y_i))$.
– ROTAC : On input (aot.transfer, sid_{AOT}, σ) from $\mathcal{V}_{\text{AOT}}$, $\mathcal{V}$ sends the message (aot.transfer.end, sid_{AOT}, Y_σ) to $\mathcal{V}_{\text{AOT}}$.

Theorem 3 *When (a subset of) the receivers are corrupt, the construction* NOTAC *and the construction* ROTAC *securely realize* $\mathcal{F}_{\text{NOTAC}}$ *and* $\mathcal{F}_{\text{ROTAC}}$ *respectively in the* $(\mathcal{F}_{\text{CRS}}^{\text{CSetup}}, \mathcal{F}_{\text{REG}}, \mathcal{F}_{\text{AA}}, \mathcal{F}_{\text{NOT}})$*-hybrid model and in the random oracle model if the zero-knowledge proof of knowledge scheme fulfills the zero-knowledge property defined in Sect. 2.3, if the* CPABE *scheme is secure and fulfills the key separation property defined in Sect. 3, and if the commitment scheme fulfills the binding property defined in Sect. 2.4. The construction* ROTAC *also operates in the* $\mathcal{F}_{\text{AOT}}$*-hybrid model.*

Proof We show by means of a series of hybrid games that the environment $\mathcal{Z}$ cannot distinguish between the real and the ideal ensemble with non-negligible probability. We denote by Pr [**Game** i] the probability that the environment distinguishes **Game** i from the real world protocol.

Game 0 This game corresponds to the execution of the real world protocol. Therefore, we have that Pr [**Game** 0] = 0.

Game 1 **Game** 1 follows **Game** 0, except that **Game** 1 replies the random oracle queries q_i as described in our simulation and computes a simulated zero-knowledge proof of knowledge π. The zero-knowledge property ensures that proofs can be simulated. Therefore, $|\Pr[\textbf{Game } 1] - \Pr[\textbf{Game } 0]| \leq \mathrm{Adv}_{\mathcal{A}}^{\mathsf{zk-zkpk}}$.

Game 2 **Game** 2 follows **Game** 1, except that **Game** 2 replaces the computation of a secret key for the entire universe of attributes using the algorithm ABEExt by the computation of a secret key for the attributes requested by the adversary using the algorithms ABEExtInit and ABEExtAtt. The key separation property of the CPABE scheme ensures indistinguishability between a key computed by ABEExt or by ABEExtInit and ABEExtAtt. Furthermore, the use of $\mathcal{F}_{\mathrm{NOT}}$ ensures that $\mathcal{A}$ receives no information about non-requested attributes. Therefore, $|\Pr[\textbf{Game } 2] - \Pr[\textbf{Game } 1]| \leq \mathrm{Adv}_{\mathcal{A}}^{\mathsf{ks-cpabe}}$.

Game 3 **Game** 3 follows **Game** 2, except that **Game** 3 aborts if the commitments $\langle com_l \rangle_{\forall\ l\ st\ a_l \in \mathbb{A}}$ and the commitments $\langle com'_l \rangle_{\forall\ l\ st\ a_l \in \mathbb{A}}$ sent by the adversary $\mathcal{A}$ are equal but the committed attributes are not equal. The binding property of the commitment scheme ensures that this happens with negligible probability. Therefore, $|\Pr[\textbf{Game } 3] - \Pr[\textbf{Game } 2]| \leq \mathrm{Adv}_{\mathcal{A}}^{\mathsf{bin-com}}$.

Game 4 **Game** 4 follows **Game** 3, except that **Game** 4 replaces the values B_i by random values. At this point, the random oracle queries Z_i are answered as described in our simulation. The security of the one time pad ensures that this change is indistinguishable. Therefore, $|\Pr[\textbf{Game } 4] - \Pr[\textbf{Game } 3]| \leq \mathrm{Adv}_{\mathcal{A}}^{\mathsf{otp}}$.

Game 5 **Game** 5 follows **Game** 4, except that **Game** 5 aborts if the adversary submits a random oracle Z_i such that Z_i is the message encrypted in the CPABE ciphertext A_i, but the adversary did not receive a secret key for a set of attributes that fulfills the policy associated to A_i. The probability that **Game** 5 aborts is bound by the following claim. □

Claim Under the security of the CPABE scheme, we have that $|\Pr[\textbf{Game } 5] - \Pr[\textbf{Game } 4]| \leq \mathrm{Adv}_{\mathcal{A}}^{\mathsf{sec-cpabe}}$.

Proof We construct an algorithm T that, given an adversary that makes the game **Game** 5 abort with non-negligible probability, breaks the security of the CPABE scheme with non-negligible probability. T interacts with the adversary and with the challenger of the CPABE security game as follows. When the challenger sends the parameters of the CPABE scheme, T employs those parameters to set the parameters of the construction NOTAC or ROTAC. (We note that T computes a simulated proof π and thus no knowledge of the master secret key of the CPABE scheme is required.) T picks a random index $i \in [1, N]$ and two random $\mathbb{Z}_{i,0}$ and $\mathbb{Z}_{i,1}$. T sends $\mathbb{Z}_{i,0}$, $\mathbb{Z}_{i,1}$ and the policy $\mathbb{P}_i$ to the challenger and receives a ciphertext A_i. To compute the rest of the ciphertexts A, T employs the parameters. When the adversary requests a secret key for a set of attributes through $\mathcal{F}_{\mathrm{NOT}}$, T sends those attributes to the challenger in order to obtain a secret key, which is sent to the adversary via $\mathcal{F}_{\mathrm{NOT}}$. If the adversary requests a secret key for a set of attributes that fulfill the target policy $\mathbb{P}_i$ then T

fails. Eventually, the adversary sends a random oracle query $Z_{i'}$ that makes **Game** 5 abort. If $i' \neq i$, T fails. Else, if $Z_{i'} = \mathbb{Z}_{i,0}$, T sends 0 as its guess to the challenger, else sends 1 as its guess to the challenger. □

The distribution of **Game** 5 is identical to our simulation.

The cases where the issuer $\mathcal{I}$ is corrupt and colludes with the sender or with some receivers are straightforward because, in our construction NOTAC and in our construction ROTAC, the issuer only employs the functionality $\mathcal{F}_{\text{AA}}$. We omit a formal description for those cases, since their proofs follow from the proof for a corrupt sender and the proof for (a subset of) corrupt receivers.

6 Instantiation and efficiency of our OT with access control protocol

6.1 Construction of committing CPABE with key separation

We propose a construction of committing CPABE based on the CPABE scheme in [3]. First we recall the scheme. Let $H : \{0,1\}^* \rightarrow \mathbb{G}$ be a hash function modelled as a random oracle.

$\mathsf{ABESetup}(1^\kappa)$ Run $\mathcal{G}(1^\kappa)$ to get a pairing group setup $\Gamma = (p, \mathbb{G}, \mathbb{G}_t, e, g)$. Pick random $(\alpha, \beta) \leftarrow \mathbb{Z}_p$ and output parameters $par = (\Gamma, h = g^\beta, U = e(g,g)^\alpha)$ and a master secret key $msk = (par, \beta, g^\alpha)$.

$\mathsf{ABEExt}(msk, \mathbb{A})$ Pick random $r \in \mathbb{Z}_p$ and, for each $j \in \mathbb{A}$, pick random $r_j \in \mathbb{Z}_p$. Compute the key as $sk_\mathbb{A} = (D = g^{(\alpha+r)/\beta}, \{D_j = g^r \cdot H(j)^{r_j}, D'_j = g^{r_j}\}_{j\in\mathbb{A}})$.

$\mathsf{ABEEnc}(par, m, \mathbb{P})$ For each node x in the tree $\mathcal{T}$ that defines $\mathbb{P}$, choose a polynomial q_x of degree $d_x = k_x - 1$, where k_x is the threshold value of x. Polynomials q_x are chosen in a top-down manner, starting from the root node R. Pick random $s \leftarrow \mathbb{Z}_p$ and set $q_R(0) = s$, and choose other d_R points randomly to define q_R completely. For any other node x, set $q_x(0) = q_{\mathsf{parent}(x)}(\mathsf{index}(x))$ and choose other d_x points randomly to define q_x completely. Let Y be the set of leaf nodes in $\mathcal{T}$. The ciphertext consists of $ct = (\mathcal{T}, \tilde{C} = m \cdot U^s, C = h^s, \{C_y = g^{q_y(0)}, C'_y = H(\mathsf{att}(y))^{q_y(0)}\}_{y\in Y})$.

$\mathsf{ABEDec}(par, ct, sk_\mathbb{A})$ Define as follows a recursive algorithm $\mathsf{ABEDecNode}(par, ct, sk_\mathbb{A}, x)$. If x is a leaf node and $i = \mathsf{att}(x)$ belongs to the set of attributes $\mathbb{A}$ associated with $sk_\mathbb{A}$, $\mathsf{ABEDecNode}(par, ct, sk_\mathbb{A}, x) = e(D_i, C_x)/e(D'_i, C'_x) = e(g,g)^{rq_x(0)}$. If $i \notin \mathbb{A}$, then $\mathsf{ABEDecNode}(par, ct, sk_\mathbb{A}, x) = \perp$.

If x is not a leaf node, then proceed as follows. For all the children z of x, call $\mathsf{ABEDecNode}(par, ct, sk_\mathbb{A}, z)$ and store the output as F_z. Let S_x be an arbitrary k_x-sized set of child nodes z such that $F_z \neq \perp$. If no such set exists $\mathsf{ABEDecNode}(par, ct, sk_\mathbb{A}, z) = \perp$. Otherwise, let $i = \mathsf{index}(z)$ and $S'_x = \{\mathsf{index}(z)\}_{z\in S_x}$, and, for $i \in \mathbb{Z}_p$ and a set S of elements in $\mathbb{Z}_p$, define the Lagrange coefficient $\Delta_{i,S}(x) = \prod_{j\in S, j\neq i} \frac{x-j}{i-j}$. Compute

$$F_x = \prod_{z\in S_x} F_z^{\Delta_{i,S'_x}(0)} = e(g,g)^{r\cdot q_x(0)}$$

$\mathsf{ABEDec}(par, ct, sk_{\mathbb{A}})$ executes $\mathsf{ABEDecNode}(par, ct, sk_{\mathbb{A}}, R)$, where R is the root node. If the tree $\mathcal{T}$ is satisfied by the set of attributes $\mathbb{A}$, $\mathsf{ABEDecNode}(par, ct, sk_{\mathbb{A}}, R)$ returns $A = e(g, g)^{rs}$. Finally, output $m = \tilde{C}/(e(C, D)/A)$.

This scheme fulfills the key separation property defined in Sect. 3. The algorithms $\mathsf{ABEExtInit}$ and $\mathsf{ABEExtAtt}$ are defined as follows.

$\mathsf{ABEExtInit}(msk, par)$ Pick random $r \in \mathbb{Z}_p$, set $\mathrm{st} = (par, r)$ and $sk'_{\mathbb{A}} = g^{(\alpha+r)/\beta}$ and output $(sk'_{\mathbb{A}}, \mathrm{st})$.

$\mathsf{ABEExtAtt}(\mathrm{st}, a)$ Pick random $r_a \in \mathbb{Z}_p$ and output $sk_a = (g^r \cdot H(a)^{r_a}, g^{r_a})$.

As can be seem, the output of $\mathsf{ABEExtInit}$ and $\mathsf{ABEExtAtt}$ is identically distributed to the output of ABEExt.

We provide this scheme with an efficient zero knowledge proof of knowledge of the statement $\mathsf{PK}\{(msk) : (par, msk) \leftarrow \mathsf{ABESetup}(1^\kappa)\}$, which is given by $\mathsf{PK}\{(\beta, g^\alpha) : h = g^\beta \ \wedge \ U = e(g, g^\alpha)\}$.

Additionally, we define algorithms $\mathsf{ABEValidSK}$ and $\mathsf{ABEValidCT}$.

$\mathsf{ABEValidSK}(par, \mathbb{A}, sk_{\mathbb{A}})$ Parse par as $(\Gamma, h = g^\beta, U = e(g, g)^\alpha)$. Conduct a check on the pairing group setup Γ. Parse $sk_{\mathbb{A}}$ as $(D, \{D_j, D'_j\}_{j\in\mathbb{A}})$ and, for all $j \in \mathbb{A}$, check whether $U \cdot e(D_j, g) = e(D'_j, H(j)) \cdot e(D, h)$ holds. If the check succeeds for all the attributes in $\mathbb{A}$, output 1, else output 0.

$\mathsf{ABEValidCT}(par, ct)$ Parse par as $(\Gamma, h = g^\beta, U = e(g, g)^\alpha)$. Conduct a check on the pairing group setup Γ and output 0 if it fails. Then parse ct as $(\mathcal{T}, \tilde{C} = m \cdot U^s, C = h^s, \{C_y = g^{q_y(0)}, C'_y = H(\mathsf{att}(y))^{q_y(0)}\}_{y\in Y})$ and, for all $y \in Y$, check if $e(C_y, H(\mathsf{att}(y))) = e(g, C'_y)$ holds and output 0 if it is not the case.

Define a recursive algorithm $\mathsf{ABECheckNode}(par, ct, x)$ as follows. If x is a leaf node, output $C_x = g^{q_x(0)}$. If not, for all the children z of x, call $\mathsf{ABECheckNode}(par, ct, x)$ and store the output as F_z. Let S_x be the set of child nodes. Let $i = \mathsf{index}(z)$ and $S'_x = \{\mathsf{index}(z)\}_{z\in S_x}$, and, for $i \in \mathbb{Z}_p$ and a set S of elements in $\mathbb{Z}_p$, define the Lagrange coefficient $\Delta_{i,S}(x) = \prod_{j\in S, j\neq i} \frac{x-j}{i-j}$. Compute

$$F_x = \prod_{z\in S_x} F_z^{\Delta_{i,S'_x}(0)} = g^{q_x(0)}.$$

Call $\mathsf{ABECheckNode}(par, ct, R)$, where R is the root node, and receive $g^{q_R(0)}$ as output. Check whether $e(g^{q_R(0)}, h) = e(g, C)$ holds and output 0 if it is not the case. Otherwise output 1.

6.1.1 Efficiency analysis of the CPABE *scheme*

Let $|\mathbb{G}|$ and $|\mathbb{G}_t|$ denote the bit lengths of elements of $\mathbb{G}$ and $\mathbb{G}_t$ respectively. Let $|\mathbb{A}|$ be the number of attributes used to compute a secret key and let $|Y|$ be the number of leaf nodes in an access tree used to compute a ciphertext. Table 1 illustrates the communication efficiency of this CPABE scheme.

Table 1 Efficiency analysis of the construction AA

PARAMETERS	$2 \cdot \lvert\mathbb{G}\rvert + 1 \cdot \lvert\mathbb{G}_t\rvert$
SECRET KEY	$(1 + 2\lvert\mathbb{A}\rvert) \cdot \lvert\mathbb{G}\rvert$
CIPHERTEXT	$1 \cdot \lvert\mathbb{G}_t\rvert + (1 + 2\lvert Y\rvert) \cdot \lvert\mathbb{G}\rvert$
ZK PROOF	$1 \cdot \lvert H\rvert + 1 \cdot \lvert\mathbb{Z}_p\rvert + 1 \cdot \lvert\mathbb{G}\rvert$

6.2 Construction of anonymous attribute authentication

We depict the construction of anonymous attribute authentication. Our construction uses the $\mathcal{F}_{\text{SMT}}$, $\mathcal{F}_{\text{ASMT}}$ and $\mathcal{F}_{\text{REG}}$-hybrid model, where parties make use of the ideal functionalities $\mathcal{F}_{\text{SMT}}$, $\mathcal{F}_{\text{ASMT}}$ and $\mathcal{F}_{\text{REG}}$ described in Sect. 4.2. It employs a signature scheme (KeyGen, Sign, VfSig) as depicted in Sect. 2.5. Additionally, it employs a zero-knowledge proof of knowledge scheme as described in Sect. 2.3.

Our construction works as follows. First, the issuer computes a signature key pair (pk, sk) and registers the public key pk with $\mathcal{F}_{\text{REG}}$. The issuer $\mathcal{I}$ signs the attributes that must be issued to a user $\mathcal{U}$ and provides $\mathcal{U}$ with the signature s. A user can prove to any party that the commitments $\langle com_l \rangle_{l=1}^{L}$ commit to attributes that were signed by the issuer. To this end, the user computes a zero-knowledge proof of knowledge of the openings of the commitments and of a signature from the issuer, such that the signed attributes equal the committed values.

Construction AA

Our construction employs the ideal functionalities $\mathcal{F}_{\text{SMT}}$, $\mathcal{F}_{\text{ASMT}}$ and $\mathcal{F}_{\text{REG}}$, a signature scheme (KeyGen, Sign, VfSig) and a zero-knowledge proof of knowledge scheme.

1. On input (aa.setup, sid, par_c, VfCom), $\mathcal{I}$ does the following:
   - Abort if $sid \neq (\mathcal{I}, sid')$.
   - Run $(pk, sk) \leftarrow \text{KeyGen}(1^k, L_{max})$. (Here, 1^k is adjusted so that pk and par_c allow the computation of zero-knowledge proofs of knowledge of equality between signed and committed attributes.)
   - Send (reg.register, sid, $\langle pk, par_c, \text{VfCom} \rangle$) to $\mathcal{F}_{\text{REG}}$.
   - Store $(pk, sk, par_c, \text{VfCom})$.
   - Output (aa.setup.end, sid).
2. On input (aa.getparams, sid), a party $\mathcal{P}$ does the following:
   - Send (reg.retrieve, sid) to $\mathcal{F}_{\text{REG}}$ and wait for a message (reg.retrieve, sid, $\langle pk, par_c, \text{VfCom} \rangle$) from $\mathcal{F}_{\text{REG}}$.
   - Store $(pk, par_c, \text{VfCom})$.
   - Output (aa.getparams.end, sid, par_c, VfCom).
3. On input (aa.issue, $sid, \mathcal{U}, \langle a_l \rangle_{l=1}^{L}$), the issuer $\mathcal{I}$ and the user $\mathcal{U}$ do the following:
   - $\mathcal{I}$ aborts if $(pk, sk, par_c, \text{VfCom})$ is not stored or if $L > L_{max}$.
   - $\mathcal{I}$ runs $s \leftarrow \text{Sign}(sk, \langle a_l \rangle_{l=1}^{L})$.
   - $\mathcal{I}$ sets $sid_{\text{SMT}} = (\mathcal{I}, \mathcal{U}, sid'')$ for a fresh sid'' and sends (smt.send, sid_{SMT}, $[\langle a_l \rangle_{l=1}^{L}, s]$) to $\mathcal{F}_{\text{SMT}}$.
   - $\mathcal{U}$ receives (smt.send.end, sid_{SMT}, $[\langle a_l \rangle_{l=1}^{L}, s]$) from $\mathcal{F}_{\text{SMT}}$.
   - If $(pk, par_c, \text{VfCom})$ is not stored, $\mathcal{U}$ executes (aa.getparams, sid).
   - $\mathcal{U}$ runs $b \leftarrow \text{VfSig}(pk, s, \langle a_l \rangle_{l=1}^{L})$.
   - If $b = 1$, $\mathcal{U}$ stores $[\langle a_l \rangle_{l=1}^{L}, s]$ and outputs (aa.issue.end, sid, $\langle a_l \rangle_{l=1}^{L}$).
4. On input (aa.prove, sid, $\mathcal{P}$, $\langle com_l, a_l, open_l \rangle_{l=1}^{L}$), a user $\mathcal{U}$ and the party $\mathcal{P}$ do the following:
   - $\mathcal{U}$ aborts if there is no tuple $[\langle a'_l \rangle_{l=1}^{L'}, s]$ stored such that $\langle a_l \rangle_{l=1}^{L} \subseteq \langle a'_l \rangle_{l=1}^{L'}$.
   - $\mathcal{U}$ aborts if $1 \neq \text{VfCom}(par_c, com_l, a_l, open_l)$ for any $l \in [1, L]$.

– $\mathcal{U}$ computes the following zero-knowledge proof of knowledge π. Let f be a mapping that takes as input the index $l \in [1, L']$ of a signed attribute a'_l and outputs the index $f(l) \in [1, L]$ of the commitment to an attribute $a_{f(l)}$ such that $a_{f(l)} = a'_l$, or outputs $\perp$ if such attribute does not exist.

$$\mathsf{PK}\{(\langle a_l \rangle_{l=1}^{L'}, \langle open_{f(l)} \rangle_{l \in [1,L'], f(l) \neq \perp}, s) :$$
$$\langle 1 \leftarrow \mathsf{VfCom}(par_c, , _{f(l)}, a_{f(l)}, open_{f(l)}) \rangle_{l \in [1,L'], f(l) \neq \perp} \wedge$$
$$1 \leftarrow \mathsf{VfSig}(pk, s, \langle a_l \rangle_{l=1}^{L'})\}$$

The first statement requires that $\langle a_{f(l)}, open_{f(l)} \rangle_{l \in [1,L'], f(l) \neq \perp}$ are valid openings of the commitments $\langle , _{f(l)} \rangle_{l \in [1,L'], f(l) \neq \perp}$. The second statement requires that the attributes $\langle a_l \rangle_{l=1}^{L'}$ are signed by $\mathcal{I}$ in the signature s.

– $\mathcal{U}$ sets $sid_{\text{ASMT}} = (\mathcal{P}, sid'')$ for a fresh sid'' and sends ($\mathsf{asmt.send}$, sid_{ASMT}, $[\langle com_l \rangle_{l=1}^{L}$, $\pi]$) to $\mathcal{F}_{\text{ASMT}}$.
– $\mathcal{P}$ receives ($\mathsf{asmt.send.end}$, sid_{ASMT}, $[\langle com_l \rangle_{l=1}^{L}, \pi]$, P) from $\mathcal{F}_{\text{ASMT}}$.
– $\mathcal{P}$ verifies π.
– If π is correct, $\mathcal{P}$ outputs ($\mathsf{aa.prove.end}$, sid, $\langle com_l \rangle_{l=1}^{L}$).

Table 2 Efficiency analysis of the construction AA

SETUP PHASE	
$\mathcal{I} \rightarrow \mathcal{U}$	$(L_{max} + 5) \cdot \lvert\mathbb{G}\rvert$
ISSUING PHASE	
$\mathcal{I} \rightarrow \mathcal{U}$	$(L' + 2) \cdot \lvert\mathbb{Z}_p\rvert + 1 \cdot \lvert\mathbb{G}\rvert$
PROOF PHASE	
$\mathcal{U} \rightarrow \mathcal{P}$	$L \cdot \lvert\mathbb{G}_p\rvert + 1 \cdot \lvert H\rvert + (L' + 6) \cdot \lvert\mathbb{Z}_p\rvert + 2 \cdot \lvert\mathbb{G}\rvert$

6.2.1 *Efficiency analysis of construction* AA

To analyse the efficiency of the construction AA, we instantiate the commitment scheme with the Pedersen commitment scheme [24], which we describe in Sect. 2.4, the signature scheme with the Au el al. [2] signature scheme, which we describe in Sect. 2.5, and the zero-knowledge proof of knowledge scheme with the Fiat-Shamir transform [13], which we describe in Sect. 2.3.

Let $\lvert\mathbb{Z}_p\rvert$, $\lvert\mathbb{G}_p\rvert$, and $\lvert\mathbb{G}\rvert$ denote the bit lengths of elements of $\mathbb{Z}_p$, $\mathbb{G}_p$, and $\mathbb{G}$ respectively. Let $\lvert H\rvert$ denote the bit length of the outputs of the hash function H. Let L_{max} be the maximum number of user attributes and L' be the number of attributes signed. Table 2 illustrates the communication efficiency of this instantiation of construction AA. We omit the use of the functionalities $\mathcal{F}_{\text{SMT}}$, $\mathcal{F}_{\text{ASMT}}$ and $\mathcal{F}_{\text{REG}}$.

6.2.2 *Security analysis of construction* AA

Theorem 4 *Construction* AA *securely realizes* $\mathcal{F}_{\text{AA}}$ *in the* $\mathcal{F}_{\text{SMT}}$, $\mathcal{F}_{\text{ASMT}}$ *and* $\mathcal{F}_{\text{REG}}$*-hybrid model if the signature scheme (*KeyGen, Sign, VfSig*) is unforgeable as defined in Sect. 2.5 and the zero-knowledge proof of knowledge scheme fulfills the properties of zero-knowledge and weak simulation extractability defined in Sect. 2.3.*

To prove that our protocol securely realizes the ideal functionality $\mathcal{F}_{\text{AA}}$, we have to show that for any environment $\mathcal{Z}$ and any adversary $\mathcal{A}$ there exists a simulator $\mathcal{V}$, such that $\mathcal{Z}$ cannot distinguish whether it is interacting with $\mathcal{A}$ and the protocol in the real world or with $\mathcal{V}$ and $\mathcal{F}_{\text{AA}}$. The simulator thereby plays the role of all honest parties in the real world and interacts with $\mathcal{F}_{\text{AA}}$ for all corrupt parties in the ideal world.

Our simulator $\mathcal{V}$ employs any simulator $\mathcal{V}_{\text{SMT}}$, $\mathcal{V}_{\text{ASMT}}$ and $\mathcal{V}_{\text{REG}}$ for the constructions that realize $\mathcal{F}_{\text{SMT}}$, $\mathcal{F}_{\text{ASMT}}$ and $\mathcal{F}_{\text{REG}}$ respectively. We note that the simulators for all the constructions that realize $\mathcal{F}_{\text{SMT}}$, $\mathcal{F}_{\text{ASMT}}$ and $\mathcal{F}_{\text{REG}}$ communicate with each of those functionalities through the same interfaces. These are the interfaces that our simulator employs to communicate with $\mathcal{V}_{\text{SMT}}$, $\mathcal{V}_{\text{ASMT}}$ and $\mathcal{V}_{\text{REG}}$. $\mathcal{V}$ forwards all the messages exchanged between $\mathcal{V}_{\text{SMT}}$, $\mathcal{V}_{\text{ASMT}}$ and $\mathcal{V}_{\text{REG}}$ and the adversary $\mathcal{A}$. When $\mathcal{A}$ sends a message that corresponds to a protocol that realizes $\mathcal{F}_{\text{SMT}}$, $\mathcal{F}_{\text{ASMT}}$ or $\mathcal{F}_{\text{REG}}$, $\mathcal{V}$ implicitly forwards that message to the respective simulator $\mathcal{V}_{\text{SMT}}$, $\mathcal{V}_{\text{ASMT}}$ or $\mathcal{V}_{\text{REG}}$.

Case $\mathcal{U}$ corrupt. We first describe the simulator $\mathcal{V}$ for the case in which a subset of the users is corrupt. We note that users can also act as verifying parties. In our simulation, we employ the simulation algorithm $\mathcal{S}$ and the extractor algorithm $\mathcal{E}$ of the non-interactive zero-knowledge proof of knowledge scheme described in Sect. 2.3.

- On input (aa.setup, sid, par_c, VfCom) from $\mathcal{F}_{\text{AA}}$, $\mathcal{V}$ runs $(pk, sk) \leftarrow \mathsf{KeyGen}(sp, L_{max})$. $\mathcal{V}$ stores $(pk, sk, par_c, \text{VfCom})$ and sends the message (reg.register, sid, $\langle pk, par_c, \text{VfCom}\rangle$) to $\mathcal{V}_{\text{REG}}$. Upon receiving the message (reg.register, sid) from $\mathcal{V}_{\text{REG}}$, $\mathcal{V}$ sends (aa.setup, sid) to $\mathcal{F}_{\text{AA}}$.
- On input the message (aa.getparams, sid, par_c, VfCom) from the functionality $\mathcal{F}_{\text{AA}}$, $\mathcal{V}$ sends the message (reg.retrieve, sid, $\langle pk, par_c, \text{VfCom}\rangle$) to $\mathcal{V}_{\text{REG}}$. After receiving the messages (reg.retrieve, sid) from $\mathcal{V}_{\text{REG}}$, $\mathcal{V}$ sends (aa.getparams, sid) to the functionality $\mathcal{F}_{\text{AA}}$.
- On input (reg.retrieve, sid) from $\mathcal{V}_{\text{REG}}$, if $sid = (\mathcal{I}, sid')$ and $(pk, sk, par_c, \text{VfCom})$ is stored, $\mathcal{V}$ sends (reg.retrieve, sid, $\langle pk, par_c, \text{VfCom}\rangle$) to $\mathcal{V}_{\text{REG}}$, else $\mathcal{V}$ sends (reg.retrieve, sid, $\perp$) to $\mathcal{V}_{\text{REG}}$. After receiving (reg.retrieve, sid) from $\mathcal{V}_{\text{REG}}$, $\mathcal{V}$ sends (reg.retrieve.end, sid, $\langle pk, par_c, \text{VfCom}\rangle$) to $\mathcal{V}_{\text{REG}}$.
- On input the message (aa.issue, $sid, \mathcal{U}$) from $\mathcal{F}_{\text{AA}}$, if $\mathcal{U}$ is honest, $\mathcal{V}$ sets $sid_{\text{SMT}} = (\mathcal{I}, \mathcal{U}, sid'')$ for a fresh sid'' and sends (smt.send, sid_{SMT}, $l(0^k)$) to $\mathcal{V}_{\text{SMT}}$, where k equals the length of the message $[\langle a_l\rangle_{l=1}^{L}, s]$. After receiving (smt.send, sid_{SMT}) from $\mathcal{V}_{\text{SMT}}$, $\mathcal{V}$ sends (aa.issue, sid) to $\mathcal{F}_{\text{AA}}$.
- On input the message (aa.issue, $sid, \mathcal{U}$) from $\mathcal{F}_{\text{AA}}$, if $\mathcal{U}$ is corrupt, $\mathcal{V}$ sends (aa.issue, sid) to $\mathcal{F}_{\text{AA}}$. After receiving (aa.issue.end, sid, $\langle a_l\rangle_{l=1}^{L}$) from $\mathcal{F}_{\text{AA}}$, $\mathcal{V}$ computes $s \leftarrow \mathsf{Sign}(sk, \langle a_l\rangle_{l=1}^{L})$, sets $sid_{\text{SMT}} = (\mathcal{I}, \mathcal{U}, sid'')$ for a fresh sid'' and sends (smt.send.end, sid_{SMT}, $[\langle a_l\rangle_{l=1}^{L}, s]$) to $\mathcal{V}_{\text{SMT}}$.
- On input (aa.prove.end, sid, $\mathcal{P}$) from $\mathcal{F}_{\text{AA}}$, if $\mathcal{P}$ is honest, $\mathcal{V}$ sets $sid_{\text{ASMT}} = (\mathcal{P}, sid'')$ for a fresh sid'' and sends (asmt.send, sid_{SMT}, $l(0^k)$) to $\mathcal{V}_{\text{ASMT}}$, where k equals the length of the message $[\langle com_l\rangle_{l=1}^{L}, \pi]$. After receiving (asmt.send, sid_{SMT}) from $\mathcal{V}_{\text{ASMT}}$, $\mathcal{V}$ sends (aa.prove.end, sid) to $\mathcal{F}_{\text{AA}}$.
- On input a random oracle query q_i from $\mathcal{A}$, $\mathcal{V}$ runs the zero knowledge simulator $(h_i, st) \leftarrow \mathcal{S}(1, st, q_i)$ and sends h_i to $\mathcal{A}$. The state st is initialized to $\emptyset$ before the first time the zero-knowledge simulator is run. The pair (q_i, h_i) is stored in $\mathcal{T}_H$.
- On input (aa.prove.end, sid, $\mathcal{P}$) from $\mathcal{F}_{\text{AA}}$, if $\mathcal{P}$ is corrupt, $\mathcal{V}$ sends (aa.prove.end, sid) to $\mathcal{F}_{\text{AA}}$. Upon receiving (aa.prove.end, sid, $\langle com_l\rangle_{l=1}^{L}$) from $\mathcal{F}_{\text{AA}}$, $\mathcal{V}$ runs the zero-knowledge simulator $(\pi, st) \leftarrow \mathcal{S}(2, st, [par_c, pk, \langle com_l\rangle_{l=1}^{L}])$ to obtain a simulated proof

π. The pair $(\langle com_l \rangle_{l=1}^{L}, \pi)$ is stored in $\mathcal{T}$. $\mathcal{V}$ sets $sid_{\text{ASMT}} = (\mathcal{P}, sid'')$ for a fresh sid'', picks random P, and sends (asmt.send.end, sid_{SMT}, $[\langle com_l \rangle_{l=1}^{L}, \pi]$, P) to $\mathcal{V}_{\text{ASMT}}$.

– On input (asmt.send.end, sid_{ASMT}, $[\langle com_l \rangle_{l=1}^{L}, \pi]$, P) from $\mathcal{V}_{\text{ASMT}}$, $\mathcal{V}$ verifies the proof π. If the proof is not correct, $\mathcal{V}$ does nothing. Otherwise $\mathcal{V}$ runs the extractor $w \leftarrow \mathcal{E}_{\mathcal{A}}([par_c, pk, \langle com_l \rangle_{l=1}^{L}], \pi; \rho, \mathcal{T}_H, \mathcal{T})$ to extract a witness $w = (\langle a'_l \rangle_{l=1}^{L'}, \langle open_l \rangle_{l=1}^{L}, s)$. If the adversary did not receive a message-signature pair $(\langle a'_l \rangle_{l=1}^{L'}, s')$, $\mathcal{V}$ aborts. Otherwise, $\mathcal{V}$ parses sid_{ASMT} as $(\mathcal{P}, sid'')$ and sends (aa.prove, sid, $\mathcal{P}$, $\langle com_l, a_l, open_l \rangle_{l=1}^{L}$) to $\mathcal{F}_{\text{AA}}$, where $\langle a_l \rangle_{l=1}^{L}$ includes all the attributes in $\langle a'_l \rangle_{l=1}^{L'}$ that were also committed in the commitments $\langle com_l \rangle_{l=1}^{L}$.

Theorem 5 *When a subset of the users is corrupt, the construction* AA *securely realizes* $\mathcal{F}_{\text{AA}}$ *in the* $\mathcal{F}_{\text{SMT}}$, $\mathcal{F}_{\text{ASMT}}$ *and* $\mathcal{F}_{\text{REG}}$*-hybrid model if the signature scheme (*KeyGen, Sign, VfSig*) is unforgeable as defined in Sect. 2.5 and the zero-knowledge proof of knowledge scheme fulfills the properties of zero-knowledge and weak simulation extractability defined in Sect. 2.3.*

Proof We show by means of a series of hybrid games that the environment $\mathcal{Z}$ cannot distinguish between the ensembles $\text{REAL}_{\text{AA},\mathcal{A},\mathcal{Z}}$ and $\text{IDEAL}_{\mathcal{F}_{\text{AA}},\mathcal{V},\mathcal{Z}}$ with non-negligible probability. We denote by Pr [**Game** i] the probability that the environment distinguishes **Game** i from the real world protocol.

Game 0 This game corresponds to the execution of the real world protocol. Therefore, we have that Pr [**Game** 0] = 0.

Game 1 **Game** 1 follows **Game** 0, except that **Game** 1 uses the zero-knowledge simulator $\mathcal{S}$ to reply to random oracle queries and to compute simulated proofs. The zero-knowledge property ensures that simulated proofs are indistinguishable from real proofs. Therefore, $|\Pr[\textbf{Game } 1] - \Pr[\textbf{Game } 0]| \leq \text{Adv}_{\mathcal{A}}^{\text{zk-zkpk}}$.

Game 2 **Game** 2 follows **Game** 1, except that **Game** 2 employs the extractor $\mathcal{E}$ to extract the witness of correct proofs sent by the adversary. The weak simulation extraction property ensures that extraction works with extraction error ν even when the adversary receives simulated proofs. Therefore, $|\Pr[\textbf{Game } 2] - \Pr[\textbf{Game } 1]| \leq \text{Adv}_{\mathcal{A}}^{\text{wse-zkpk}}$.

Game 3 **Game** 3 follows **Game** 2, except that, when the issuer $\mathcal{I}$ issues the attributes $\langle a_l \rangle_{l=1}^{L}$ to a corrupt user, **Game** 3 appends a new entry $[\langle a_l \rangle_{l=1}^{L}]$ to a table Tbl_a. This change does not alter the view of the environment. Therefore, Pr [**Game** 3] = Pr [**Game** 2].

Game 4 **Game** 4 follows **Game** 3, except that, when the adversary sends a correct proof π, after running the extractor $\mathcal{E}$ to extract the witness $(\langle a'_l \rangle_{l=1}^{L'}, \langle open_l \rangle_{l=1}^{L}, s)$, $\mathcal{V}$ aborts if there is no entry $[\langle a'_l \rangle_{l=1}^{L'}]$ in Tbl_a. □

Claim Under the unforgeability property of the signature scheme, we have that $|\Pr[\textbf{Game } 4] - \Pr[\textbf{Game } 3]| \leq \text{Adv}_{\mathcal{A}}^{\text{unf-sig}}$.

Proof We construct an algorithm B that, given an adversary $\mathcal{A}$ that makes **Game** 4 abort with non-negligible probability ν, breaks the unforgeability property of the signature scheme with non-negligible probability ν. B works as follows. When the challenger of the unforgeability property sends the public key pk, B employs pk to set the issuer's public key. When the issuer issues the attributes $\langle a_l \rangle_{l=1}^{L'}$ to the adversary $\mathcal{A}$, B queries the signing oracle $\mathcal{O}_s$ on input $\langle a_l \rangle_{l=1}^{L'}$ in order to obtain a signature s, which is sent to $\mathcal{A}$. For the rest of interactions with $\mathcal{A}$, B proceeds as in **Game** 4. When $\mathcal{A}$ inputs a proof π that makes **Game** 4 abort, B gets

the extracted witness $(\langle a'_l \rangle_{l=1}^{L'}, \langle open_l \rangle_{l=1}^{L}, s)$ and sends $(\langle a'_l \rangle_{l=1}^{L'}, s)$ to break the unforgeability property of the signature scheme. □

The distribution of **Game** 4 is identical to our simulation.

Case $\mathcal{I}$ and $\mathcal{U}$ corrupt. Now we describe the simulator $\mathcal{V}$ for the case in which a subset of the users and the issuer are corrupt. In our simulation, we employ the simulation algorithm $\mathcal{S}$ and the extractor algorithm $\mathcal{E}$ of the non-interactive zero-knowledge proof of knowledge scheme described in Sect. 2.3.

- On input the message (reg.register, sid, pk, par_c, VfCom) from $\mathcal{V}_{\text{REG}}$, $\mathcal{V}$ sends the message (reg.register, sid, pk, par_c, VfCom) to $\mathcal{V}_{\text{REG}}$. After receiving the message (reg.register, sid) from $\mathcal{V}_{\text{REG}}$, $\mathcal{V}$ stores (pk, par_c, VfCom). $\mathcal{V}$ sends the message (aa.setup, sid, par_c, VfCom) to $\mathcal{F}_{\text{AA}}$. After receiving the message (aa.setup, sid, par_c, VfCom) from $\mathcal{F}_{\text{AA}}$, $\mathcal{V}$ sends the message (aa.setup, sid) to $\mathcal{F}_{\text{AA}}$. After receiving (aa.setup.end, sid) from $\mathcal{F}_{\text{AA}}$, $\mathcal{V}$ sends (reg.register.end, sid) to $\mathcal{V}_{\text{REG}}$.
- On input (aa.getparams, sid, par_c, VfCom) from $\mathcal{F}_{\text{AA}}$, $\mathcal{V}$ proceeds as in the case where the issuer is honest.
- On input (reg.retrieve, sid) from $\mathcal{V}_{\text{REG}}$, $\mathcal{V}$ proceeds as in the case where the issuer is honest.
- On input (smt.send, sid_{SMT}, $[\langle a_l \rangle_{l=1}^{L}, s]$) from $\mathcal{V}_{\text{SMT}}$, $\mathcal{V}$ runs $b \leftarrow \mathsf{VfSig}(pk, s, \langle a_l \rangle_{l=1}^{L})$. If $b = 0$, $\mathcal{V}$ does nothing. Else, $\mathcal{V}$ parses sid_{SMT} as $(\mathcal{I}, \mathcal{U}, sid'')$ and sends (aa.issue, $sid, \mathcal{U}$, $\langle a_l \rangle_{l=1}^{L}$) to $\mathcal{F}_{\text{AA}}$. After receiving (aa.issue, $sid, \mathcal{U}$) from $\mathcal{F}_{\text{AA}}$, $\mathcal{V}$ sends (aa.issue, sid) to $\mathcal{F}_{\text{AA}}$.
- On input (aa.prove.end, sid, $\mathcal{P}$) from $\mathcal{F}_{\text{AA}}$, if $\mathcal{P}$ is honest, $\mathcal{V}$ proceeds as in the case where the issuer is honest.
- On input a random oracle query q_i from $\mathcal{A}$, $\mathcal{V}$ proceeds as in the case where the issuer is honest.
- On input (aa.prove.end, sid, $\mathcal{P}$) from $\mathcal{F}_{\text{AA}}$, if $\mathcal{P}$ is corrupt, $\mathcal{V}$ proceeds as in the case where the issuer is honest.
- On input (asmt.send.end, sid_{ASMT}, $[\langle com_l \rangle_{l=1}^{L}, \pi], P)$ from $\mathcal{V}_{\text{ASMT}}$, $\mathcal{V}$ verifies the proof π. If the proof is not correct, $\mathcal{V}$ does nothing. Otherwise $\mathcal{V}$ runs the extractor $w \leftarrow \mathcal{E}_{\mathcal{A}}([par_c, pk, \langle com_l \rangle_{l=1}^{L}], \pi; \rho, \mathcal{T}_H, \mathcal{T})$ to extract a witness $w = (\langle a'_l \rangle_{l=1}^{L'}, \langle open_l \rangle_{l=1}^{L}, s)$. $\mathcal{V}$ parses sid_{ASMT} as $(\mathcal{P}, sid'')$ and sends (aa.prove, sid, $\mathcal{P}$, $\langle com_l, a_l, open_l \rangle_{l=1}^{L}$) to $\mathcal{F}_{\text{AA}}$, where $\langle a_l \rangle_{l=1}^{L}$ includes all the attributes in $\langle a'_l \rangle_{l=1}^{L'}$ that were also committed in the commitments $\langle com_l \rangle_{l=1}^{L}$.

Theorem 6 *When a subset of the users and the issuer are corrupt, the construction* AA *securely realizes $\mathcal{F}_{\text{AA}}$ in the $\mathcal{F}_{\text{SMT}}$, $\mathcal{F}_{\text{ASMT}}$ and $\mathcal{F}_{\text{REG}}$-hybrid model if the zero-knowledge proof of knowledge scheme fulfills the properties of zero-knowledge and weak simulation extractability defined in Sect. 2.3.*

Game 0 *This game corresponds to the execution of the real world protocol. Therefore, we have that Pr* [***Game*** 0] = 0.

Game 1 ***Game*** 1 *follows* ***Game*** 0*, except that* ***Game*** 1 *uses the zero-knowledge simulator $\mathcal{S}$ to reply to random oracle queries and to compute simulated proofs. The zero-knowledge property ensures that simulated proofs are indistinguishable from real proofs. Therefore,* $|Pr\,[\textbf{Game}\ 1] - Pr\,[\textbf{Game}\ 0]| \leq Adv_{\mathcal{A}}^{\mathsf{zk-zkpk}}$.

Game 2 ***Game*** 2 *follows* ***Game*** 1*, except that* ***Game*** 2 *employs the extractor $\mathcal{E}$ to extract the witness of correct proofs sent by the adversary. The weak simulation extraction property ensures that extraction works with extraction error ν even when the*

adversary receives simulated proofs. Therefore, $|Pr\,[\textbf{Game}\ 2] - Pr\,[\textbf{Game}\ 1]| \leq Adv_{\mathcal{A}}^{\mathsf{wse-zkpk}}$.

The distribution of **Game** 2 is identical to our simulation.

6.3 Construction of non-adaptive oblivious transfer

We depict the construction NOT of non-adaptive oblivious transfer. Our construction uses the $\mathcal{F}_{\text{ASMT}}$-hybrid model, where parties make use of the ideal functionality $\mathcal{F}_{\text{ASMT}}$ described in Sect. 4.2. It also employs a non-interactive zero-knowledge proof of knowledge scheme.

Our construction is based on the OT protocol by Camenisch et al. [9]. It modifies the protocol in [9] in order to introduce commitments to selection values, which are sent from the receiver to the sender.

The construction NOT is executed by a sender $\mathcal{E}$ and a receiver $\mathcal{R}$. These parties are activated through the not.initrec, not.init, not.request and not.transfer interfaces.

1. The receiver $\mathcal{R}$ is activated through the not.initrec interface on input the sender identity $\mathcal{E}$, which is included in the session identifier. $\mathcal{R}$ sends an initialization message to the sender $\mathcal{E}$ by using $\mathcal{F}_{\text{ASMT}}$.
2. The sender $\mathcal{E}$ is activated through the not.init interface on input the messages $(m_1, \ldots, m_N)$ and the parameters of the commitment scheme. The sender $\mathcal{E}$ encrypts the messages as follows. First, $\mathcal{E}$ computes signatures X_i on the indices $i \in [1, N]$. Then, $\mathcal{E}$ encrypts the messages by computing $E_i = e(h, X_i) \cdot m_i$, where h is the sender's secret. Finally, $\mathcal{E}$ stores the ciphertexts $F_i = (E_i, X_i)$. The sender $\mathcal{E}$ sends the encryption parameters, the commitment parameters and the ciphertexts to the receiver.[2]
3. The receiver $\mathcal{R}$ is activated through the not.request interface on input $\langle com_k, \sigma_k, open_k \rangle_{k=1}^{K}$. For each selection value σ_k, the receiver sends to the sender the commitment com_k along with a blinded version V_{σ_k} of the signature X_{σ_k} and a zero-knowledge proof of knowledge that the selection value in the commitment and in the signature are equal.
4. The sender $\mathcal{E}$ is activated through the not.transfer interface on input the commitments $\langle com_k \rangle_{k=1}^{K}$. For each commitment com_k, the sender sends to the receiver a value W_{σ_k} that allows the receiver to decrypt the message encrypted in E_σ. The sender also sends a proof that this value is correctly computed.

Now we describe formally the construction NOT.

Construction NOT

Our construction is parameterized with a number of messages N and a message space $\mathcal{M}$ and employs the ideal functionality $\mathcal{F}_{\text{ASMT}}$ and a zero-knowledge proof of knowledge scheme.

1. On input (not.initrec, *sid*), $\mathcal{R}$ does the following:
   - Abort if $sid \neq (\mathcal{E}, sid')$ or if $(\Gamma, H, pk_{OT}, F_1, \ldots, F_N, \pi_1, par_c, \mathsf{VfCom})$ is already stored.
   - $\mathcal{R}$ sets $sid_{\text{ASMT}} = (\mathcal{E}, sid'')$ for a fresh sid'' and sends (asmt.send, sid_{ASMT}, initialization) to $\mathcal{F}_{\text{ASMT}}$.
   - $\mathcal{E}$ receives (asmt.send.end, sid_{ASMT}, initialization, P) from $\mathcal{F}_{\text{ASMT}}$.
   - $\mathcal{E}$ stores (sid_{ASMT}, P).
   - $\mathcal{E}$ outputs (not.initrec.end, *sid*).

[2] This construction encrypts messages in $\mathbb{G}_t$. As explained in [9], it is possible to hash the message into $\mathbb{G}_t$ or to extract a random pad from the element in the target group and use $\oplus$ to encrypt the message.

2. On input (not.init, sid, $m_1, \ldots, m_N$, par_c, VfCom), $\mathcal{E}$ does the following:
   - Abort if (sid_{ASMT}, P) is not stored, or if (Γ, h, H, pk_{OT}, $F_1, \ldots, F_N, \pi_1, par_c$, VfCom) is already stored, or if, for $i = 1$ to N, $m_i \notin \mathcal{M}$.
   - Run $\Gamma = (p, \mathbb{G}, \mathbb{G}_t, e, g) \leftarrow \mathcal{G}(1^k)$. (Here, 1^k is adjusted so that Γ and par_c allow the computation of zero-knowledge proofs of knowledge of equality between committed messages and elements of $\mathbb{Z}_p$.)
   - Pick random $h \leftarrow \mathbb{G}$ and compute $H = e(g, h)$.
   - Pick random $x_{OT} \leftarrow \mathbb{Z}_p$ and compute $pk_{OT} = g^{x_{OT}}$.
   - For $i = 1$ to N, do the following:
     - Compute $X_i = g^{1/(x_{OT}+i)}$ and encrypt $E_i = e(h, X_i) \cdot m_i$.
     - Set $F_i = (E_i, X_i)$.
   - Compute a zero-knowledge proof of knowledge $\pi_1 = \mathsf{PK}\{(h) : H = e(h, g)\}$.
   - Store (Γ, h, H, pk_{OT}, $F_1, \ldots, F_N, \pi_1, par_c$, VfCom).
   - $\mathcal{E}$ sends (asmt.reply, sid_{ASMT}, $\langle \Gamma, H, pk_{OT}, F_1, \ldots, F_N, \pi_1, par_c$, VfCom$\rangle$, P) to $\mathcal{F}_{\text{ASMT}}$.
   - $\mathcal{R}$ receives the message (asmt.reply.end, sid_{ASMT}, $\langle \Gamma, H, pk_{OT}, F_1, \ldots, F_N, \pi_1, par_c$, VfCom$\rangle$) from $\mathcal{F}_{\text{ASMT}}$.
   - For $i = 1$ to N, $\mathcal{R}$ does the following:
     - Parse $F_i = (E_i, X_i)$.
     - Abort if $e(X_i, pk_{OT} g^i) \neq e(g, g)$.
   - Store (Γ, H, pk_{OT}, $F_1, \ldots, F_N, \pi_1, par_c$, VfCom) and output (not.init.end, sid, par_c, VfCom).
3. On input (not.request, sid, $\langle com_k, \sigma_k, open_k \rangle_{k=1}^K$), $\mathcal{R}$ and $\mathcal{E}$ run the following protocol:
   - $\mathcal{R}$ aborts if $1 \neq$ VfCom$(par_c, com_k, \sigma_k, open_k)$ for any $k \in [1, K]$ or if $[sid_{\text{ASMT}}, \langle V_k, v_k, com_k, \sigma_k, open_k \rangle_{k=1}^K]$ is already stored.
   - For $k = 1$ to K, $\mathcal{R}$ picks random $v_k \leftarrow \mathbb{Z}_p$ and computes $V_k = X_{\sigma_k}^{v_k}$.
   - $\mathcal{R}$ computes a zero-knowledge proof of knowledge $\pi_2 = \mathsf{PK}\{(\langle \sigma_k, v_k, open_k \rangle_{k=1}^K) : \wedge_{k=1}^K e(V_k, pk_{OT}) = e(V_k, g)^{-\sigma_k} e(g, g)^{v_k} \wedge_{k=1}^K 1 =$ VfCom$(par_c, com_k, \sigma_k, open_k)\}$.
   - $\mathcal{R}$ sets $sid_{\text{ASMT}} = (\mathcal{E}, sid'')$ for a fresh sid'', stores $[sid_{\text{ASMT}}, \langle V_k, v_k, com_k, \sigma_k, open_k \rangle_{k=1}^K]$, and sends (asmt.send, sid_{ASMT}, $[\langle com_k, V_k \rangle_{k=1}^K, \pi_2]$) to $\mathcal{F}_{\text{ASMT}}$.
   - $\mathcal{R}$ receives the message (asmt.send.end, sid_{ASMT}, $\langle [\langle com_k, V_k \rangle_{k=1}^K, \pi_2] \rangle$, P) from $\mathcal{F}_{\text{ASMT}}$.
   - $\mathcal{E}$ verifies the proof π_2 and aborts if it is not correct.
   - $\mathcal{E}$ stores ($\langle com_k, V_k \rangle_{k=1}^K, \pi_2, sid_{\text{ASMT}}, P$) and outputs (not.request.end, sid, com).
4. On input (not.transfer, sid, $\langle com_k \rangle_{k=1}^K$), $\mathcal{E}$ and $\mathcal{R}$ run the following protocol:
   - $\mathcal{E}$ aborts if there is not tuple ($\langle com'_k, V_k \rangle_{k=1}^K, \pi_2, sid_{\text{ASMT}}, P$) such that $com'_k = com_k$ (for all $k \in [1, K]$) stored.
   - $\mathcal{E}$ computes $W_k = e(h, V_k)$ for all $k \in [1, K]$.
   - $\mathcal{E}$ computes a zero-knowledge proof of knowledge $\pi_3 = \mathsf{PK}\{(h) : H = e(h, g) \bigwedge_{k=1}^K W_k = e(h, V_k)\}$.
   - $\mathcal{E}$ sends (asmt.reply, sid_{ASMT}, $[\langle W \rangle_{k=1}^K, \pi_3]$, P) to $\mathcal{F}_{\text{ASMT}}$.
   - $\mathcal{R}$ receives (asmt.reply.end, sid_{ASMT}, $[\langle W \rangle_{k=1}^K, \pi_3]$) from $\mathcal{F}_{\text{ASMT}}$.
   - $\mathcal{R}$ aborts if there is not tuple $[sid_{\text{ASMT}}, \langle V_k, v_k, com_k, \sigma_k, open_k \rangle_{k=1}^K]$ stored or if the proof π_3 is not correct.
   - $\mathcal{R}$ computes $m_{\sigma_k} = E_{\sigma_k} / W_k^{1/v_k}$ for all $k \in [1, K]$ and outputs the message (not.transfer.end, sid, $\langle \sigma_k, m_{\sigma_k} \rangle_{k=1}^K$).

6.3.1 *Efficiency analysis of construction* NOT

To analyse the efficiency of the construction NOT, we instantiate the commitment scheme with the Pedersen commitment scheme [24], which we describe in Sect. 2.4, and the zero-

Table 3 Efficiency analysis of the construction NOT

INIT	
$\mathcal{E} \to \mathcal{R}$	$(3+N)\cdot\lvert\mathbb{G}\rvert + (1+N)\cdot\lvert\mathbb{G}_t\rvert + 1\cdot\lvert H\rvert$
REQ	
$\mathcal{R} \to \mathcal{E}$	$K\cdot\lvert\mathbb{G}\rvert + K\cdot\lvert\mathbb{G}_p\rvert + 3K\cdot\lvert\mathbb{Z}_p\rvert + 1\cdot\lvert H\rvert$
TRANS	
$\mathcal{E} \to \mathcal{R}$	$K\cdot\lvert\mathbb{G}\rvert + K\cdot\lvert\mathbb{G}_t\rvert + 1\cdot\lvert H\rvert$

knowledge proof of knowledge scheme with the Fiat-Shamir transform [13], which we describe in Sect. 2.3.

Let $|\mathbb{Z}_p|$, $|\mathbb{G}_p|$, $|\mathbb{G}|$ and $|\mathbb{G}_t|$ denote the bit lengths of elements of $\mathbb{Z}_p$, $\mathbb{G}_p$, $\mathbb{G}$ and $\mathbb{G}_t$ respectively. Let $|H|$ denote the bit length of the outputs of the hash function H. Let N be the number of messages and K be the number of transfer phases. Table 3 illustrates the communication efficiency of this instantiation of construction NOT. We omit the use of the functionality $\mathcal{F}_{\text{ASMT}}$.

6.3.2 Security analysis of construction NOT

Theorem 7 *The construction* NOT *securely realizes* $\mathcal{F}_{\text{NOT}}$ *in the* $\mathcal{F}_{\text{ASMT}}$*-hybrid model if the zero-knowledge proof of knowledge scheme fulfills the properties of zero-knowledge and weak simulation extractability defined in Sect. 2.3, and if the strong Diffie Hellman and power decisional Diffie Hellman assumptions as defined in [9] hold.*

To prove that the construction NOT securely realizes the ideal functionality $\mathcal{F}_{\text{NOT}}$, we have to show that for any environment $\mathcal{Z}$ and any adversary $\mathcal{A}$ there exists a simulator $\mathcal{V}$, such that $\mathcal{Z}$ cannot distinguish whether it is interacting with $\mathcal{A}$ and the protocol in the real world or with $\mathcal{V}$ and $\mathcal{F}_{\text{NOT}}$. The simulator thereby plays the role of all honest parties in the real world and interacts with $\mathcal{F}_{\text{NOT}}$ for all corrupt parties in the ideal world.

Our simulator $\mathcal{V}$ employs any simulator $\mathcal{V}_{\text{ASMT}}$ for the constructions that realizes $\mathcal{F}_{\text{ASMT}}$ respectively. We note that the simulators for all the constructions that realize $\mathcal{F}_{\text{ASMT}}$ communicate with each of those functionalities through the same interfaces. These are the interfaces that our simulator employs to communicate with $\mathcal{V}_{\text{ASMT}}$. $\mathcal{V}$ forwards all the messages exchanged between $\mathcal{V}_{\text{ASMT}}$ and the adversary $\mathcal{A}$. When $\mathcal{A}$ sends a message that corresponds to a protocol that realizes $\mathcal{F}_{\text{ASMT}}$, $\mathcal{V}$ implicitly forwards that message to the respective simulator $\mathcal{V}_{\text{ASMT}}$.

Case $\mathcal{E}$ corrupt. We first describe the simulator $\mathcal{V}$ for the case in which the sender is corrupt. In our simulation, we employ the simulation algorithm $\mathcal{S}^2$ for proof of knowledge π_2 and the extractor algorithm $\mathcal{E}^1$ for proof of knowledge π_1 of the non-interactive zero-knowledge proof of knowledge scheme described in Sect. 2.3.

- On input (not.initrec, *sid*) from $\mathcal{F}_{\text{NOT}}$, the simulator $\mathcal{V}$ parses *sid* as $(\mathcal{E}, sid')$, sets $sid_{\text{ASMT}} = (\mathcal{E}, sid'')$ for a fresh sid'', and sends (asmt.send, sid_{ASMT}, $l(0^k)$) to $\mathcal{V}_{\text{ASMT}}$, where k is the length of the message initialization. After receiving (asmt.send, sid_{ASMT}) from $\mathcal{V}_{\text{ASMT}}$, $\mathcal{V}$ sends (not.initrec, *sid*) to $\mathcal{F}_{\text{NOT}}$.
- On input a random oracle query q_i from $\mathcal{A}$, $\mathcal{V}$ runs the zero knowledge simulator $(h_i, st) \leftarrow \mathcal{S}(1, st, q_i)$ and sends h_i to $\mathcal{A}$. The state st is initialized to $\emptyset$ before the first time the zero-knowledge simulator is run. The pair (q_i, h_i) is stored in $\mathcal{T}_H$. The zero

knowledge simulator that answers random oracle queries works the same way for proofs π_1, π_2 and π_3.

- On input (asmt.reply, sid_{ASMT}, $\langle \Gamma, H, pk_{OT}, F_1, \ldots, F_N, \pi_1, par_c, \text{VfCom} \rangle$, P) from $\mathcal{V}_{\text{ASMT}}$, $\mathcal{V}$ verifies the proof π_1. For $i = 1$ to N, $\mathcal{V}$ parses F_i as (E_i, X_i) and verifies the signature X_i as in the real world protocol. If any of these verification does not succeed, $\mathcal{V}$ does nothing. Otherwise $\mathcal{V}$ sets $\mathcal{T}$ to empty and runs the extractor $w \leftarrow \mathcal{E}^1_{\mathcal{A}}(\langle \Gamma, H \rangle, \pi_1; \rho, \mathcal{T}_H, \mathcal{T})$ to extract a witness $w = h$. For $i = 1$ to N, $\mathcal{V}$ decrypts the messages by running $m_i = E_i / e(h, X_i)$. $\mathcal{V}$ sends (not.init, sid, $m_1, \ldots, m_N, par_c$, VfCom) to $\mathcal{F}_{\text{NOT}}$. Upon receiving (not.init, sid, par_c, VfCom) from $\mathcal{F}_{\text{NOT}}$, $\mathcal{V}$ sends (not.init, sid) to $\mathcal{F}_{\text{NOT}}$.
- On input (not.request, sid) from $\mathcal{F}_{\text{NOT}}$, send (not.request, sid) from $\mathcal{V}$. After receiving (not.request.end, sid, $\langle com_k \rangle_{k=1}^K$) from $\mathcal{F}_{\text{NOT}}$, $\mathcal{V}$ picks a random $V_k \leftarrow \mathbb{G}$ for $k \in [1, K]$ and computes a simulated proof $(\pi_2, st) \leftarrow \mathcal{S}^2(2, st, [\Gamma, par_c, pk_{OT}, \langle V_k, com_k \rangle_{k=1}^K])$. $\mathcal{V}$ sets $sid_{\text{ASMT}} = (\mathcal{E}, sid'')$ for a fresh sid'', picks random P, stores $[sid_{\text{ASMT}}, \langle V_k, com_k, \rangle_{k=1}^K]$ and sends (asmt.send.end, sid_{ASMT}, $[\langle com_k, V_k \rangle_{k=1}^K, \pi_2], P)$ to $\mathcal{V}_{\text{ASMT}}$.
- On input (asmt.reply, sid_{ASMT}, $[\langle W \rangle_{k=1}^K, \pi_3], P)$ from $\mathcal{V}_{\text{ASMT}}$, $\mathcal{V}$ does nothing if there is no tuple $[sid_{\text{ASMT}}, \langle V_k, com_k, \rangle_{k=1}^K]$ stored or if the proof π_3 is not correct. Otherwise $\mathcal{V}$ sends (not.transfer, sid, $\langle com_k \rangle_{k=1}^K$) to $\mathcal{F}_{\text{NOT}}$. Upon receiving (not.transfer, sid) from $\mathcal{F}_{\text{NOT}}$, $\mathcal{V}$ sends (not.transfer, sid) to $\mathcal{F}_{\text{NOT}}$.

Theorem 8 *When the sender is corrupt, the construction* NOT *securely realizes the functionality $\mathcal{F}_{\text{NOT}}$ in the $\mathcal{F}_{\text{ASMT}}$-hybrid model if the zero knowledge proof of knowledge scheme for π_2 is zero-knowledge and the zero knowledge proof of knowledge scheme for π_1 fulfills the extractability property, which is implied by the weak simulation extractability property defined in Sect. 2.3.*

Proof We give a proof sketch which follows the proof in [9].

Game 0 This game corresponds to the execution of the real world protocol. Therefore, we have that $\Pr[\textbf{Game } 0] = 0$.

Game 1 **Game** 1 follows **Game** 0, except that **Game** 1 uses the zero-knowledge simulator to reply to random oracle queries and to compute the simulated proofs π_2. The zero-knowledge property ensures that simulated proofs are indistinguishable from real proofs. Therefore, $|\Pr[\textbf{Game } 1] - \Pr[\textbf{Game } 0]| \leq \text{Adv}^{\text{zk-zkpk}}_{\mathcal{A}}$.

Game 2 **Game** 2 follows **Game** 1, except that **Game** 2 employs the extractor to extract the witness of the proof π_1 send by the adversary. The extraction property ensures that extraction works with extraction error ν. Therefore, $|\Pr[\textbf{Game } 2] - \Pr[\textbf{Game } 1]| \leq \text{Adv}^{\text{wse-zkpk}}_{\mathcal{A}}$. □

The distribution of **Game** 2 is identical to our simulation.

Case $\mathcal{R}$ corrupt. Now we describe the simulator $\mathcal{V}$ for the case in which the receiver is corrupt. In our simulation, we employ the simulation algorithms $\mathcal{S}^1$ and $\mathcal{S}^3$ for the proofs of knowledge π_1 and π_3 and the extractor algorithm $\mathcal{E}^2$ for proof of knowledge π_2 of the non-interactive zero-knowledge proof of knowledge scheme described in Sect. 2.3.

- On input (asmt.send, sid_{ASMT}, initialization) from $\mathcal{V}_{\text{ASMT}}$, $\mathcal{V}$ sends (not.initrec, sid) to $\mathcal{F}_{\text{NOT}}$. After receiving (not.initrec, sid) from $\mathcal{F}_{\text{NOT}}$, $\mathcal{V}$ sends (asmt.send, sid_{ASMT}, $l(0^k)$) to $\mathcal{V}_{\text{ASMT}}$, where k is the length of the message initialization. After receiving (asmt.send, sid_{ASMT}) from $\mathcal{V}_{\text{ASMT}}$, $\mathcal{V}$ sends (not.initrec, sid) to $\mathcal{F}_{\text{NOT}}$.

- On input a random oracle query q_i from $\mathcal{A}$, $\mathcal{V}$ runs the zero knowledge simulator $(h_i, st) \leftarrow \mathcal{S}(1, st, q_i)$ and sends h_i to $\mathcal{A}$. The state st is initialized to $\emptyset$ before the first time the zero-knowledge simulator is run. The pair (q_i, h_i) is stored in $\mathcal{T}_H$. The zero knowledge simulator that answers random oracle queries works the same way for proofs π_1, π_2 and π_3.
- On input (not.init, sid, par_c, VfCom) from $\mathcal{F}_{\text{NOT}}$, $\mathcal{V}$ sends (not.init, sid) to $\mathcal{F}_{\text{NOT}}$. Upon receiving (not.init.end, sid, par_c, VfCom) from $\mathcal{F}_{\text{NOT}}$, $\mathcal{V}$ runs $\Gamma = (p, \mathbb{G}, \mathbb{G}_t, e, g) \leftarrow \mathcal{G}(1^k)$, picks random $h \leftarrow \mathbb{G}$ and computes $H = e(g, h)$. $\mathcal{V}$ picks random $x_{OT} \leftarrow \mathbb{Z}_p$ and computes $pk_{OT} = g^{x_{OT}}$. For $i = 1$ to N, $\mathcal{V}$ computes $X_i = g^{1/(x_{OT}+i)}$, picks random $E_i \leftarrow \mathbb{G}$ and sets $F_i = (E_i, X_i)$. $\mathcal{V}$ computes a simulated proof $(\pi_1, st) \leftarrow \mathcal{S}^1(2, st, \langle \Gamma, H \rangle)$. $\mathcal{V}$ sends (asmt.reply.end, sid_{ASMT}, $\langle \Gamma, H, pk_{OT}, F_1, \ldots, F_N, \pi_1, par_c$, VfCom$\rangle$) to $\mathcal{V}_{\text{ASMT}}$.
- On input (asmt.send, sid_{ASMT}, $[\langle com_k, V_k \rangle_{k=1}^K, \pi_2]$) from $\mathcal{V}_{\text{ASMT}}$, $\mathcal{V}$ does nothing if the proof π_2 is not correct. Otherwise $\mathcal{V}$ stores $(sid_{\text{ASMT}}, \langle com_k, V_k \rangle_{k=1}^K)$, sets $\mathcal{T}$ to empty and runs the extractor $w \leftarrow \mathcal{E}_{\mathcal{A}}^2([\Gamma, par_c, pk_{OT}, \langle com_k, V_k \rangle_{k=1}^K], \pi_2; \rho, \mathcal{T}_H, \mathcal{T})$ to extract a witness $w = \langle \sigma_k, v_k, open_k \rangle_{k=1}^K$. $\mathcal{V}$ aborts if $\sigma_k \notin [1, N]$. Otherwise $\mathcal{V}$ stores $w = \langle \sigma_k, v_k, open_k \rangle_{k=1}^K$ and sends (not.request, sid, $\langle com_k, \sigma_k, open_k \rangle_{k=1}^K$) to $\mathcal{F}_{\text{NOT}}$.
- On input (not.transfer.end, sid, $\langle com_{\sigma_k}, m_{\sigma_k} \rangle_{k=1}^K$) from $\mathcal{F}_{\text{NOT}}$, $\mathcal{V}$ employs the extracted witness $w = \langle \sigma_k, v_k, open_k \rangle_{k=1}^K$ to compute $W_k = (E_{\sigma_k}/m_k)^{v_k}$ for $k = 1$ to K. $\mathcal{V}$ computes a simulated proof $(\pi_3, st) \leftarrow \mathcal{S}^3(2, st, [\Gamma, H, \langle W_k \rangle_{k=1}^K])$. $\mathcal{V}$ gets the sid_{ASMT} stored along with $\langle , _{\sigma_k} \rangle_{k=1}^K$ and sends (asmt.reply.end, sid_{ASMT}, $[\langle W \rangle_{k=1}^K, \pi_3]$) to $\mathcal{V}_{\text{ASMT}}$.

Theorem 9 *When the receiver is corrupt, the constructions* NOT *securely realizes* $\mathcal{F}_{\text{NOT}}$ *in the* $\mathcal{F}_{\text{ASMT}}$*-hybrid model if the zero knowledge proof of knowledge scheme for* π_1 *and for* p_3 *is zero-knowledge and the zero knowledge proof of knowledge scheme for* π_2 *fulfills the extractability property, which is implied by the weak simulation extractability property defined in Sect. 2.3. Additionally, the strong Diffie Hellman and power decisional Diffie Hellman assumptions must hold.*

Proof We give a proof sketch which follows the proof in [9].

Game 0 This game corresponds to the execution of the real world protocol. Therefore, we have that $\Pr[\textbf{Game } 0] = 0$.

Game 1 **Game** 1 follows **Game** 0, except that the simulator extracts the witness from proof of knowledge π_2. The extraction property ensures that extraction works with extraction error ν. Therefore, $|\Pr[\textbf{Game } 1] - \Pr[\textbf{Game } 0]| \leq \text{Adv}_{\mathcal{A}}^{\text{wse-zkpk}}$.

Game 2 **Game** 2 follows **Game** 1, except that **Game** 2 aborts if $\sigma \notin [1, N]$. The probability that **Game** 2 aborts is bound by the strong Diffie Hellman assumption. We refer to [9] for the proof of this claim. Therefore, $|\Pr[\textbf{Game } 2] - \Pr[\textbf{Game } 1]| \leq \text{Adv}_{\mathcal{A}}^{\text{SDH}}$.

Game 3 **Game** 3 follows **Game** 2, except that **Game** 3 simulates the proof π_1 and π_3 and computes the value W as in our simulation. The zero-knowledge property ensures that simulated proofs are indistinguishable from real proofs. Therefore, $|\Pr[\textbf{Game } 3] - \Pr[\textbf{Game } 2]| \leq \text{Adv}_{\mathcal{A}}^{\text{zk-zkpk}}$.

Game 4 **Game** 4 follows **Game** 3, except that **Game** 4 replaces the values E computed in the initialization phase by random values. The probability of distinguishing between **Game** 4 and **Game** 3 is negligible under the power decisional Diffie Hellman assumption. We refer to [9] for the proof of this claim. Therefore, $|\Pr[\textbf{Game } 4] - \Pr[\textbf{Game } 3]| \leq \text{Adv}_{\mathcal{A}}^{\text{PDDH}}$. □

The distribution of **Game** 4 is identical to our simulation.

Table 4 Efficiency analysis of the construction AOT

INIT	
$\mathcal{E} \to \mathcal{R}$	$(3+N)\cdot\lvert\mathbb{G}\rvert + (1+N)\cdot\lvert\mathbb{G}_t\rvert + 1\cdot\lvert H\rvert$
TRANS	
$\mathcal{R} \to \mathcal{E}$	$K\cdot\lvert\mathbb{G}\rvert + 2K\cdot\lvert\mathbb{Z}_p\rvert + K\cdot\lvert H\rvert$
$\mathcal{E} \to \mathcal{R}$	$K\cdot\lvert\mathbb{G}\rvert + K\cdot\lvert\mathbb{G}_t\rvert + K\cdot\lvert H\rvert$

6.4 Construction of adaptive oblivious transfer

To realize $\mathcal{F}_{\text{AOT}}$, we employ the construction in [9]. The main difference between $\mathcal{F}_{\text{AOT}}$ and the ideal functionality in [9] is that $\mathcal{F}_{\text{AOT}}$ interacts with more than one receiver, who remain anonymous towards the sender. In order to adapt the construction in [9] to realize $\mathcal{F}_{\text{AOT}}$, we do the following changes. First, the initialization message that the sender sends to the receiver in [9] is instead registered with $\mathcal{F}_{\text{REG}}$. The receivers retrieve this message from $\mathcal{F}_{\text{REG}}$. In the transfer phase, the request message that the receiver sends to the sender and the response message that the sender sends to the receiver are communicated through $\mathcal{F}_{\text{ASMT}}$. We omit a formal description of the construction and the security analysis because it essentially follows the construction and security analysis given in Sect. 6.3 for the non-adaptive OT construction.

6.4.1 Efficiency analysis of construction AOT

To analyse the efficiency of the construction AOT, we instantiate the zero knowledge proof of knowledge scheme with the Fiat-Shamir transform [13], which we describe in Sect. 2.3. Let $|\mathbb{Z}_p|$, $|\mathbb{G}|$ and $|\mathbb{G}_t|$ denote the bit lengths of elements of $\mathbb{Z}_p$, $\mathbb{G}$ and $\mathbb{G}_t$ respectively. Let $|H|$ denote the bit length of the outputs of the hash function H. Let N be the number of messages and K be the number of transfer phases. Table 4 illustrates the communication efficiency of this instantiation of construction AOT. We omit the use of the functionalities $\mathcal{F}_{\text{REG}}$ and $\mathcal{F}_{\text{ASMT}}$.

6.5 Efficiency analysis of our OT with access control protocol

To analyze the efficiency of the constructions NOTAC and ROTAC, we instantiate the CPABE scheme and the functionalities $\mathcal{F}_{\text{AA}}$, $\mathcal{F}_{\text{NOT}}$ and $\mathcal{F}_{\text{AOT}}$ with the constructions described in Sect. 6. The cost of the functionality $\mathcal{F}_{\text{CRS}}^{\text{CSetup}}$ is ommitted.

Let $|\mathbb{Z}_p|$, $|\mathbb{G}_p|$, $|\mathbb{G}|$ and $|\mathbb{G}_t|$ denote the bit lengths of elements of $\mathbb{Z}_p$, $\mathbb{G}_p$, $\mathbb{G}$ and $\mathbb{G}_t$ respectively. Let $|H|$ denote the bit length of the outputs of the hash function H. Let N be the number of messages. Let $|\mathbb{A}|$ be the number of attributes issued and used to compute a secret key and let $|Y|$ be the average number of leaf nodes in an access tree used to compute a ciphertext. Finally, let $|m|$ be the length of a message and k be the security parameter.

Table 5 illustrates the communication efficiency of this instantiation of construction NOTAC. The cost of the issuing phase reflects the cost of one issuing phase between the issuer and a receiver. The cost of the transfer phase reflects the cost of one transfer phase where the receiver inputs $|\mathbb{A}|$ attributes. As can be seen, in the construction NOTAC the cost does not depend on the number of messages transferred. In contrast, in the construction ROTAC, the cost of executing one instance of $\mathcal{F}_{\text{AOT}}$ must be added to the overall cost.

Table 5 Efficiency analysis of the construction NOTAC

INIT	
$\mathcal{E} \to \mathcal{F}_{\text{REG}}$	$1 \cdot \lvert H\rvert + N(1 + 2\lvert Y\rvert) \cdot \lvert\mathbb{G}\rvert + (N + 1) \cdot \lvert\mathbb{G}_t\rvert + 1 \cdot \lvert\mathbb{Z}_p\rvert + N(\lvert m\rvert + k)$
GET	
$\mathcal{F}_{\text{REG}} \to \mathcal{R}$	$1 \cdot \lvert H\rvert + N(1 + 2\lvert Y\rvert) \cdot \lvert\mathbb{G}\rvert + (N + 1) \cdot \lvert\mathbb{G}_t\rvert + 1 \cdot \lvert\mathbb{Z}_p\rvert + N(\lvert m\rvert + k)$
ISSUE	
$\mathcal{I} \to \mathcal{R}$	$(\lvert\mathbb{A}\rvert + 2) \cdot \lvert\mathbb{Z}_p\rvert + 1 \cdot \lvert\mathbb{G}\rvert$
TRANS	
$\mathcal{E} \to \mathcal{R}$	$(4 + L_{max}) \cdot \lvert\mathbb{G}\rvert + (2 + L_{max}) \cdot \lvert\mathbb{G}_t\rvert + 1 \cdot \lvert H\rvert$
$\mathcal{R} \to \mathcal{E}$	$(\lvert\mathbb{A}\rvert + 2) \cdot \lvert\mathbb{G}\rvert + 2\lvert\mathbb{A}\rvert \cdot \lvert\mathbb{G}_p\rvert + (4\lvert\mathbb{A}\rvert + 6) \cdot \lvert\mathbb{Z}_p\rvert + 2 \cdot \lvert H\rvert$
$\mathcal{E} \to \mathcal{R}$	$\lvert\mathbb{A}\rvert \cdot \lvert\mathbb{G}\rvert + \lvert\mathbb{A}\rvert \cdot \lvert\mathbb{G}_t\rvert + 1 \cdot \lvert H\rvert$

7 Conclusion

We have proposed a non-restricted and a restricted OTAC scheme. Our constructions are based on any committing CPABE with key separation scheme. We define the committing and key separation properties for CPABE. We also provide a construction based on the CPABE scheme in [3], which supports any access control policy that can be described by a monotone access structure. In comparison to previous work, this allows our OTAC scheme to support efficiently a wider class of access control policies.

We also describe a blind key extraction with access control protocol for any committing CPABE with key separation scheme. Our protocol works in a hybrid model that employs two novel ideal functionalities for non-adaptive OT and for anonymous attribute authentication. Unlike existing non-adaptive OT functionalities, the novel functionalities allow the sender of the OTAC scheme to check whether an input to the non-adaptive OT functionality equals an input to the anonymous attribute authentication functionality.

Acknowledgments This work was supported by the European Commission's Seventh Framework Programme under the FutureID project (Agreement #318424).

References

1. Abe M., Camenisch J., Dubovitskaya M., Nishimaki R.: Universally composable adaptive oblivious transfer (with access control) from standard assumptions. In: Proceedings of the 2013 ACM Workshop on Digital Identity Management, pp. 1–12. ACM, Berlin (2013).
2. Au M.H., Susilo W., Mu Y.: Constant-size dynamic k-taa. In: Prisco R.D., Yung, M. (eds.) SCN. Lecture Notes in Computer Science, vol. 4116, pp. 111–125. Springer, Heidelberg (2006).
3. Bethencourt J., Sahai A., Waters B.: Ciphertext-policy attribute-based encryption. In: IEEE Symposium on Security and Privacy, pp. 321–334. IEEE Computer Society, Washington, DC (2007).
4. Camenisch J., Stadler M.: Proof systems for general statements about discrete logarithms. Technical Report TR 260. Institute for Theoretical Computer Science, ETH Zürich (1997).
5. Camenisch J., Dubovitskaya M., Enderlein R.R., Neven G.: Oblivious transfer with hidden access control from attribute-based encryption. In: Visconti I., Prisco R.D. (eds.) SCN. Lecture Notes in Computer Science, vol. 7485, pp. 559–579. Springer, Berlin (2012).
6. Camenisch J., Dubovitskaya M., Neven G.: Oblivious transfer with access control. In: Al-Shaer E., Jha S., Keromytis A.D. (eds.) ACM Conference on Computer and Communications Security, pp. 131–140. ACM, New York (2009).

7. Camenisch J., Dubovitskaya M., Neven G., Zaverucha G.M.: Oblivious transfer with hidden access control policies. In: Catalano D., Fazio N., Gennaro R., Nicolosi A. (eds.) Public Key Cryptography. Lecture Notes in Computer Science, vol. 6571, pp. 192–209. Springer, Berlin (2011).
8. Camenisch J., Lehmann A., Neven G., Rial A.: Privacy-preserving auditing for attribute-based credentials. In: Computer Security—ESORICS 2014, pp. 109–127. Springer, Cham (2014).
9. Camenisch, J., Neven, G., Shelat, A.: Simulatable adaptive oblivious transfer. In: Naor (ed.) Advances in Cryptology—EUROCRYPT 2007. Proceedings 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Barcelona, Spain, 20–24, May 2007. Lecture Notes in Computer Science, vol. 4515, pp. 573–590. Springer, Heidelberg (2007).
10. Canetti R.: Universally composable security: a new paradigm for cryptographic protocols. In: FOCS, pp. 136–145. IEEE Computer Society, Washington, DC (2001).
11. Coull S.E., Green M., Hohenberger S.: Controlling access to an oblivious database using stateful anonymous credentials. In: Jarecki S., Tsudik G. (eds.) Public Key Cryptography. Lecture Notes in Computer Science, vol. 5443, pp. 501–520. Springer, Heidelberg (2009).
12. Faust S., Kohlweiss M., Marson G.A., Venturi D.: On the non-malleability of the fiat-shamir transform. In: Progress in Cryptology—INDOCRYPT 2012, pp. 60–79. Springer, Heidelberg (2012).
13. Fiat A., Shamir A.: How to prove yourself: practical solutions to identification and signature problems. In: Odlyzko A.M. (ed.) CRYPTO '86. Lecture Notes in Computer Science, vol. 263, pp. 186–194. Springer, Heidelberg (1987).
14. Goldwasser S., Micali S., Rivest R.L.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM J. Comput. **17**, 281–308 (1988).
15. Green M., Hohenberger S.: Blind identity-based encryption and simulatable oblivious transfer. In: ASIACRYPT, pp. 265–282 (2007).
16. Green M., Hohenberger S.: Blind identity-based encryption and simulatable oblivious transfer. In: Advances in Cryptology–ASIACRYPT 2007, pp. 265–282. Springer, Berlin (2007).
17. Guleria V., Dutta R.: Issuer-free adaptive oblivious transfer with access policy. In: Information Security and Cryptology—ICISC 2014, pp. 402–418. Springer, Berlin (2014).
18. Guleria V., Dutta R.: Universally composable identity based adaptive oblivious transfer with access control. In: Information Security and Cryptology, pp. 109–129. Springer, Beijing (2014).
19. Han J., Susilo W., Mu Y., Yan J.: Attribute-based oblivious access control. Comput. J. **55**, 1202–1215 (2012).
20. Han J., Susilo W., Mu Y., Yan J.: Efficient oblivious transfers with access control. Comput. Math. Appl. **63**, 827–837 (2012).
21. Kohlweiss M., Faust S., Fritsch L., Gedrojc B., Preneel B.: Efficient oblivious augmented maps: location-based services with a payment broker. In: Borisov N., Golle P. (eds.) Privacy Enhancing Technologies. Lecture Notes in Computer Science, vol. 4776, pp. 77–94. Springer, Berlin (2007).
22. Naor M., Pinkas B.: Oblivious transfer with adaptive queries. In: Wiener M.J. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 1666, pp. 573–590. Springer, Heidelberg (1999).
23. Nishide T., Yoneyama K., Ohta K.: Attribute-based encryption with partially hidden encryptor-specified access structures. In: Bellovin S.M., Gennaro R., Keromytis A.D., Yung M. (eds.) ACNS. Lecture Notes in Computer Science, vol. 5037, pp. 111–129 (2008).
24. Pedersen T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum J. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 576, pp. 129–140. Springer, Heidelberg (1991).
25. Rabin M.O.: How to Exchange Secrets by Oblivious Transfer. Harvard Aiken Computation Laboratory, Cambridge (1981).
26. Rial A., Preneel B.: Blind attribute-based encryption and oblivious transfer with fine-grained access control. COSIC Technical Report (2010).
27. Sahai A., Waters B.: Fuzzy identity-based encryption. In: Advances in Cryptology—EUROCRYPT 2005. pp. 457–473. Springer, New York (2005).
28. Waters B.: Ciphertext-policy attribute-based encryption: an expressive, efficient, and provably secure realization. In: Public Key Cryptography—PKC 2011. pp. 53–70. Springer, Heidelberg (2011).
29. Xu L., Zhang F.: Oblivious transfer with complex attribute-based access control. In: Information Security and Cryptology—ICISC 2010. pp. 370–395. Springer, Beijing (2011).
30. Xu L.L., Zhang F.G.: Oblivious transfer with threshold access control. J. Inf. Sci. Eng. **28**, 555–570 (2012).
31. Xu L., Zhang F., Wen Y.: Oblivious transfer with access control and identity-based encryption with anonymous key issuing. J. Electron. (China) **28**, 571–579 (2011).

32. Zhang Y., Au M.H., Wong D.S., Huang Q., Mamoulis N., Cheung D.W., Yiu S.M.: Oblivious transfer with access control: realizing disjunction without duplication. In: Joye M., Miyaji A., Otsuka A. (eds.) Pairing. Lecture Notes in Computer Science, vol. 6487, pp. 96–115. Springer, Berlin (2010).