Access Controls for Oblivious and Anonymous Systems

Scott E. Coull[†] Matthew D. Green[‡] Susan Hohenberger[‡]

[†]University of North Carolina
Chapel Hill, NC
scoull@cs.unc.edu

[‡]Johns Hopkins University
Baltimore, MD
{mgreen,susan}@cs.jhu.edu

Abstract

The use of privacy-enhancing cryptographic protocols, such as anonymous credentials and oblivious transfer, often has a detrimental effect on the ability of providers to effectively implement access controls on their content. In this paper, we propose a *stateful anonymous credential* system that allows the provider to implement non-trivial, real-world access controls on oblivious protocols conducted with anonymous users. Our stateful anonymous credential system models the behavior of users as a state machine, and embeds that state within an anonymous credential to restrict access to resources based on the state information. The use of state machine models of user behavior allows the provider to restrict the users' actions according to a wide variety of access control models without learning anything about the users' identities or actions. Our system is secure in the standard model under basic assumptions, and, after an initial setup phase, each transaction requires only constant time. As a concrete example, we show how to implement the Brewer-Nash (Chinese Wall) and Bell-La Padula (Multilevel Security) access control models within our credential system. Furthermore, we combine our credential system with a simulatable, adaptive oblivious transfer scheme to create a privacy-friendly oblivious database with strong access controls.

1 Introduction

The increasing use of distributed computing services (*i.e.*, "cloud computing") raises significant concerns about the privacy of sensitive information such as financial records or medical data. While encryption can protect the *contents* of an outsourced database, the service provider still collects sensitive information about which users access the data, and how they access it. Researchers have already taken steps to address this problem, with the development of efficient protocols for Private Information Retrieval and Oblivious Transfer. Unfortunately, using these protocols requires a significant tradeoff: without knowing who is accessing which data, a service provider cannot implement basic security mechanisms, such as authentication, access controls, and auditing. What is really needed are systems which preserve the user's privacy, to the extent possible, while allowing the provider to maintain control over his database.

Medical databases provide a particularly compelling example of this tradeoff. Through her queries or those of her physicians, a patient may reveal sensitive information about her medical condition to the service provider. Simultaneously, the service provider is bound by laws, such as HIPAA, to authenticate users, audit their activities, and control access to these highly sensitive medical records. The recent introduction of outsourced medical record services, like Google Health [37] and Microsoft HealthVault [42], make this a pressing problem.

Cryptographic systems, such as anonymous credentials and conditional oblivious transfer, offer partial resolution to this user/provider conflict. To address the service provider's need for authentication and the users' desire for privacy, anonymous credentials [22, 15] can be used to ensure that only authorized personnel gain access to the system without revealing the identity of those users. Oblivious transfer [43, 19] and private information retrieval [24] systems allow the user to retrieve data in such a way that the provider learns nothing about the user's query or the data being retrieved. The combination of these two systems leads to privacy-friendly databases *when the user gets her choice of any files with no restrictions.* Unfortunately, that scenario rules out many practical database applications. Worse, the previous work in this area provides no insight as to how access control might ever be incorporated into such a database, since traditional access control mechanisms assume knowledge of the items being requested.

Contributions. The goal of this work is to create an efficient mechanism for controlling access to a wide variety of anonymous and oblivious systems, while simultaneously maintaining all of the privacy guarantees of those systems. Moreover, these privacy-preserving access controls must conform to real-world access control models that are used by traditional computer systems. To do so, we model the actions of users as a state machine, and introduce *stateful anonymous credentials* to embed the user's state within her credentials. Our stateful anonymous credentials are built on top of well-known blind signature protocols [41, 15, 16, 5], and our construction is secure in the standard model under basic assumptions, such as Strong RSA. In addition, we show how to combine our system with anonymous and oblivious protocols, and provide a concrete instance by constructing an oblivious database with access controls using the oblivious transfer scheme of Camenisch, Neven, and shelat [19]. We also introduce a technique for efficiently proving that a committed value lies in a hidden range that is unknown to the verifier, which may be of independent interest.

Since our stateful credential system uses a state machine model of user behavior, we are able to implement a wide range of non-trivial history-dependent access control models, including the Biba [3], Clark-Wilson [25], Bell-La Padula [2], and Brewer-Nash [8] models. These access control models are used in a number of settings, including financial institutions and classified government systems. To provide concrete examples of real-world access controls within our stateful credential system, we show how to implement discretionary access controls, as well as the Bell-La Padula [2] and Brewer-Nash [8] access control models.

Related Work. Previous work on conditional oblivious transfer and anonymous credentials has sought to limit user actions while simultaneously maintaining privacy. Techniques from e-cash and anonymous credential systems have been used to place simple limitations on an anonymous user's actions, primarily related to limiting the number of uses of a credential. These limitations include preventing a user from logging in more than once in a given time period [11], authenticating anonymously at most k times [48], or preventing a user from exchanging too much money with a single merchant [13]. As alluded to earlier, the limitations provided by anonymous credential systems are not general enough to encode the real-world access controls necessary for practical use. Aiello, Ishai, and Reingold [1] proposed priced oblivious transfer, in which each user is given a declining balance that can be spent on each transfer. However, here user anonymity is not protected, and the protocol is also vulnerable to selective-failure attacks in which a malicious server induces faults to deduce the user's selections [43, 19]. The more general concept of conditional oblivious transfer was proposed by Di Crescenzo, Ostrovsky, and Rajagopolan [28] and subsequently strengthened by Blake and Kolesnikov [4]. In conditional oblivious transfer, the sender and receiver maintain private inputs (x and y, respectively) to some publicly known predicate $q(\cdot, \cdot)$ (*e.g.,* the greater than equal to relation on integers). The items in the oblivious transfer scheme are encrypted such that the receiver can complete the oblivious transfer and recover her data if and only if $q(x, y) = 1$. Rather than providing a specific type of limitation or restricting the limitation to a particular protocol, our proposed system instead provides a general method by which arbitrary access control policies can be added to a wide variety of anonymous and oblivious protocols.

Subsequent to our own work, Camenisch, Dubovitskaya, and Neven [10] proposed a combination of the Camenisch, Neven, and shelat oblivious transfer scheme [19] with Boneh-Boyen signatures [5] to apply *stateless* access controls to oblivious transfer protocols. Specifically, their system embeds a fixed list of "categories" that a user has access to within a Boneh-Boyen signature, and uses zero-knowledge proof techniques to ensure that the oblivious transfer query belongs to one of the allowed "categories" before completing the protocol. Since their access controls are stateless, they do not require the user to update her credential. However, this naturally limits the scope of the access controls that can be applied within their system to simple discretionary access policies, which are a strict subset of the policies enabled by our proposed scheme. In fact, several extensions to our system exist to not only provide general stateful access control models, but also very efficient stateless access controls under standard cryptographic assumptions.

2 Technical Preliminaries

In this section, we recall some basic building blocks, and then introduce a new primitive, *hidden range proofs*, which may be of independent interest. In order to provide the most general construction of our stateful anonymous credential system, we defer the instantiations of these building blocks until we combine our credential system with specific cryptographic protocols in Section 4. For the remainder of the paper, we denote the security parameter as 1^κ.

Bilinear Groups. Let $\mathsf{BMsetup}$ be an algorithm that, on input 1^κ, outputs the parameters for a bilinear mapping as $\gamma = (p, \mathbb{G}, \mathbb{G}_T, e, g \in \mathbb{G})$, where g generates $\mathbb{G}$, the groups $\mathbb{G}, \mathbb{G}_T$ each have prime order p, and a bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ with the following properties:

1. Bilinear: $\forall g_1, g_2 \in \mathbb{G}$ and $x, y \in \mathbb{Z}_p$, $e(g_1^x, g_2^y) = e(g_1, g_2)^{xy}$.
2. Non-degenerate: $e(g, g) \neq 1$.
3. Computable: There is an efficient algorithm to compute $e(g_1, g_2) \forall g_1, g_2 \in \mathbb{G}$.

Commitments. A commitment scheme allows a user to bind herself to a chosen value without revealing that value to the recipient of the commitment. This commitment to the value ensures that the user cannot change her choice (i.e., *binding*), while simultaneously ensuring that the recipient of the commitment does not learn anything about the value it contains (i.e., *hiding*). In general, commitment schemes consist of a setup algorithm CSetup, a commitment algorithm Commit, and an opening algorithm $\mathsf{Decommit}$, which we describe here:

1. $\mathsf{CSetup}(1^\kappa)$. On input a security parameter, outputs the commitment parameters ρ.
2. $\mathsf{Commit}(v_1, \dots, v_m; r)$. On input of values $v_1, \dots, v_m$ and randomness r, outputs a commitment C.
3. $\mathsf{Decommit}(C)$. On input of a commitment C, outputs an opening D for the values in the commitment.

Pedersen [46] provides a perfectly hiding and computationally binding commitment scheme for prime order groups, and Fujisaki and Okamoto [33] provide a similar (though, statistically hiding) commitment scheme for composite order groups.

Zero-Knowledge Protocols. In a zero-knowledge protocol [35] a user proves a statement to another party without revealing anything about the statement other than its veracity. We use several standard results for proving statements about discrete logarithms, such as (1) a proof of knowledge of a discrete logarithm modulo a prime [47] or a composite [33, 29], (2) a proof of knowledge of equality of representation modulo two (possibly different) prime [23] or composite [18] moduli, (3) a proof that a commitment opens to the product of two other committed values [17, 20, 7], and (4) a proof of the disjunction or conjunction of any two of the previous [27]. Additionally, we make use of techniques to prove equality of discrete logarithms from different groups [23, 18].

Signatures with Efficient Protocols. In order to construct our anonymous credential system, we make use of signature schemes that have the following properties: (1) the signature scheme should be secure against existential forgery attacks in the adaptive chosen message attack model [36], (2) there exists an efficient protocol for a user to obtain a signature on the value(s) in a commitment [46, 33] without the signer learning anything about the value(s), and (3) a zero-knowledge protocol for proving knowledge of the values within a signature. The signature scheme consists of the algorithms $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$, which we describe below:

1. $\mathsf{KeyGen}(1^\kappa)$. On input a security parameter, outputs a keypair (pk, sk).
2. $\mathsf{Sign}(sk, M_1, \ldots, M_n)$. On input one or more messages and a secret signing key, outputs the signature σ.
3. $\mathsf{Verify}(pk, \sigma, M_1, \ldots, M_n)$. On input a signature, message(s) and public verification key, outputs 1 if the signature verifies, 0 otherwise.

Camenisch and Lysyanskaya designed two such schemes, one that is secure under the Strong RSA assumption in composite order groups [15] and another using bilinear groups under the LRSW assumption [16]. In addition, Boneh and Boyen also provide a signature scheme that maintains these properties, which is secure under the Strong Diffie-Hellman assumption in bilinear groups [5].

Since our anonymous credential scheme could be combined with oblivious transfer protocols, we also require that the signature scheme meet the definition of *selective-failure blindness*, as defined by Camenisch, Neven, and shelat [19]. The definition of selective-failure blindness consists of a game where the adversary outputs a public key and two messages. The adversary is then given black-box access to two instances of the signing algorithm, $\mathsf{Sign}(M_b)$ and $\mathsf{Sign}(M_{1-b})$ for $b \xleftarrow{\$} \{0, 1\}$, where the adversary acts as the signer for her chosen public key. The output from these algorithm instances is s_b and s_{1-b}, respectively. If $s_b \neq \perp$ and $s_{1-b} \neq \perp$, then the adversary receives (s_0, s_1). When $s_b = \perp$ or $s_{1-b} = \perp$, the adversary receives $(\perp, \epsilon)$ or $(\epsilon, \perp)$, respectively. If both $s_b = \perp$ and $s_{1-b} = \perp$, then the adversary gets $(\perp, \perp)$. The signatures scheme is considered to be selective-failure blind if no p.p.t. adversary has non-negligible advantage in guessing the bit b. The original constructions of Camenisch and Lysyanskaya [15, 16], and Boneh and Boyen [5] were not proved secure under this stronger definition of blindness. However, recent work by Fischlin and Schröder [32] has shown that it is possible to efficiently convert any blind signature scheme, including those mentioned above, into a selective-failure blind scheme through the addition of only a single commitment.

Pseudorandom Functions. A pseudorandom function [34] is an efficiently computable function whose output cannot be distinguished (with non-negligible advantage) from random elements by a computationally bounded adversary. We denote the pseudorandom function as $f_k(\cdot)$, where k is a randomly chosen key. The Dodis-Yampolskiy PRF (DY PRF) [31] is secure under the y-Decisional Diffie-Hellman Inversion assumption for a polynomial sized input space within prime order groups. Recently, Jarecki and Liu have shown that the DY PRF is also secure in composite order groups [39]. Naor and Reingold describe another PRF that is secure under the Decisional Diffie-Hellman assumption in prime and composite order groups [44].

Verifiable Encryption. A verifiable encryption scheme is an encryption mechanism that allows the encryptor to prove that a ciphertext contains a message without revealing the message to the verifier. Specifically, a verifiable encryption scheme takes as input a public key pk and a message m, and produces a ciphertext $C = \mathsf{Encrypt}_{pk}(m)$ and a commitment $A = \mathsf{Commit}(m)$. The verifier accepts the ciphertext C if the encryptor provides a valid zero-knowledge proof that the value contained in the commitment is the same as that which is contained in the ciphertext. The protocol ensures that the verifier accepts incorrect encryptions with only negligible probability and, as long as the underlying encryption mechanism is semantically secure, that the verifier does not learn anything about the message. Techniques presented by Camenisch and Damgård allow any semantically secure encryption scheme to be turned into a verifiable encryption scheme [9].

Hidden-Range Proofs. Standard techniques [21, 17, 17, 6] allow us to efficiently prove that a committed value lies in a *public* integer interval (*i.e.,* where the interval is known to both the prover and verifier). In our protocols, it is useful to *hide* this interval from the verifier, and instead have the prover show that a committed value lies between the openings of two other commitments.

Fortunately, this can be done efficiently as follows. Suppose we wish to show that $a \leq j \leq b$, for positive numbers a, j, b without revealing them. This is equivalent to showing that $0 \leq (j - a)$ and $0 \leq (b - j)$. We only need to get these two sums reliably into commitments, and can then employ the standard techniques since the range (≥ 0) is now public. Using a group $\mathbf{G} = \langle \mathbf{g} \rangle$, where n is a special RSA modulus, $\mathbf{g}$ is a quadratic residue modulo n and $\mathbf{h} \in \mathbf{G}$. The prover commits to these values as $A = \mathbf{g}^a\mathbf{h}^{r_a}$, $J = \mathbf{g}^j\mathbf{h}^{r_j}$, and $B = \mathbf{g}^b\mathbf{h}^{r_b}$, for random values $r_a, r_j, r_b \in \{0,1\}^\ell$ where ℓ is a security parameter. The verifier next computes a commitment to $(j - a)$ as J/A and to $(b - j)$ as B/J. The prover and verifier then proceed with the standard public interval proofs with respect to these commitments, which for technical reasons require groups where Strong RSA holds.

3 Stateful Anonymous Credentials

The goal of typical anonymous credential systems is to provide users with a way of proving certain attributes about themselves (*e.g.,* age, or height) without revealing their identity. Users conduct this proof by obtaining a credential from a credential provider, and subsequently "showing" the credential without revealing their identity. In addition to the properties of typical credentials, a stateful anonymous credential system adds the notion of credential state, which is embedded as an attribute within the credential. The user may update the state in her credential according to some well-defined *policy* describing her allowed use of the credential as dictated by the credential provider. In practice, this may limit the user to a finite number of states, or a particular sequential ordering of states. To maintain the user's anonymity, it is important that the update protocol not leak information about the credential's current state beyond what the user chooses to reveal.

At a high level, the stateful anonymous credential system, which is defined by the tuple of algorithms ($\mathsf{Setup}, \mathsf{ObtainCred}, \mathsf{UpdateCred}, \mathsf{ProveCred}$), operates as follows. First, the user and credential provider negotiate the use of a specified policy using the $\mathsf{ObtainCred}$ protocol. The negotiated policy determines the way in which the user will be allowed to update her credential. After the protocol completes, the user receives an anonymous credential that embeds her initial state in the policy, in addition to other attributes. Next, the user can prove (in zero-knowledge) that the credential she holds embeds a given state, or attribute, just as she would in other anonymous credential systems by using the $\mathsf{ProveCred}$ protocol. This allows the user anonymous access to services, while the entity checking the credential is assured of the user's attributes, as well as her state in the specified policy. These proofs can be done in such a way that the verifying entity learns nothing about the user's state or attributes. Finally, when the user wishes to update her credential to reflect a change in her state, she interacts with the credential provider using the UpdateCred protocol, to prove (again, in zero-knowledge) her current state and the existence of a transition in the policy from her current state to her intended next state. As with the ProveCred protocol, the provider learns nothing about the user other than the fact that her state change is allowed by the policy previously negotiated within the ObtainCred protocol.

Policy Model. To represent the policies for our stateful credential system, we use directed graphs, which can be thought of as a state machine that describes the user's behavior over time. We describe the *policy graph* Π_{pid} as the set of *tags* of the form $(\mathsf{pid}, S \rightarrow T)$, where pid is the identity of the policy and $S \rightarrow T$ represents a directed edge from state S to state T. Thus, the user's credential embeds the identity of the policy pid and the user's current state in the policy graph. When the user updates her credential, she chooses a tag and then proves that the policy id she is following is the same as what is provided in the tag and the tag encodes an edge from her current state to her desired next state.

While these policy graphs are rather simplistic, they can represent complicated policies. For instance, a policy graph can encode the user's history with respect to accessing certain resources up to the largest cycle in the graph. Moreover, we can extend the policy graph tags to include auxiliary information about the actions that the user is allowed to perform at each state. By doing so, we allow the graph to dynamically control the user's access to various resources according to her behavior and history, as well as other attributes. In Section 4.1, we examine how to extend these policy graphs to provide non-trivial, real-world access control policies for a variety of anonymous and oblivious cryptographic protocols.

These policy graphs can be created in such a way that the users may reach a terminal state, and therefore would be unable to continue updating (and consequently using) their credential. In this case, it may be possible for an adversary to perform traffic analysis to infer the policy that the user is following. To prevent this, we consider the use of *null transitions* in the graph. The null transitions occur as self-loops on the terminal states of the policy graph, and allow the user to update her credential as often as she wishes to prevent such traffic analysis attacks. However, the updates performed on these credentials only allow the user access to a predefined null resource. The specifics of this null resource are dependent on the cryptographic protocol that the credential system is coupled with, and we describe their implementation for them in oblivious databases in Section 4.

3.1 Protocol Descriptions and Definitions for Stateful Credentials

A stateful anonymous credential scheme consists of the four protocols: Setup, ObtainCred, UpdateCred, and ProveCred. These protocols occur between a user $\mathcal{U}$ and credential provider $\mathcal{P}$. While we only use a two-party model, with the credential provider playing the roles of both provider and verifier, there is no restriction to extending our system to a three party model with a separate verifier. We will now describe their input/output behavior and intended functionality.

1. $\mathsf{Setup}(\mathcal{U}(1^k), \mathcal{P}(1^k, \Pi_1, \ldots, \Pi_n))$**:** The provider $\mathcal{P}$ generates parameters *params* and a keypair $(pk_{\mathcal{P}}, sk_{\mathcal{P}})$ for the credential scheme. For each graph Π_i to be enforced, $\mathcal{P}$ also generates a cryptographic representation $\Pi_i^{\mathcal{C}}$ and publishes this value via an authenticated channel. Each user $\mathcal{U}$ generates a keypair and requests that it be certified by a trusted CA.

2. $\mathsf{ObtainCred}(\mathcal{U}(pk_{\mathcal{P}}, sk_{\mathcal{U}}, \Pi_i^{\mathcal{C}}), \mathcal{P}(pk_{\mathcal{U}}, sk_{\mathcal{P}}, \Pi_i^{\mathcal{C}}, S))$**:** $\mathcal{U}$ identifies herself to $\mathcal{P}$ and then receives her credential Cred which binds her to a policy graph Π and starting state S.
3. $\mathsf{UpdateCred}(\mathcal{U}(pk_{\mathcal{P}}, sk_{\mathcal{U}}, \mathsf{Cred}, T), \mathcal{P}(sk_{\mathcal{P}}, D))$**:** $\mathcal{U}$ and $\mathcal{P}$ interact such that Cred is updated from its current state to state T, but only if this transition is permitted by the policy Π_i. Simultaneously, $\mathcal{P}$ should not learn $\mathcal{U}$'s identity, attributes, or current state. To prevent replay attacks, $\mathcal{P}$ maintains a database D, which it updates as a result of the protocol.
4. $\mathsf{ProveCred}(\mathcal{U}(pk_{\mathcal{P}}, sk_{\mathcal{U}}, \mathsf{Cred}), \mathcal{P}(pk_{\mathcal{P}}, E))$**:** $\mathcal{U}$ proves possession of a credential Cred in a particular state. To prevent re-use of credentials, $\mathcal{P}$ maintains a database E, which it updates as a result of the protocol.

Security Definitions. Security definitions for anonymous credentials have traditionally been game-based. Unfortunately, the existing definitions may be insufficient for the applications considered in this work, as these definitions do not necessarily capture *correctness*. This can lead to problems when we integrate our credential system with certain cryptographic protocols, such as oblivious transfer (*c.f.,* [43, 19]). To capture the security requirements needed for our applications, we instead use a simulation-based definition, in which security of our protocols is analyzed with respect to an "ideal world" instantiation. We do not require security under concurrent executions, but rather restrict our analysis to atomic, sequential execution of each protocol. We do so because our constructions, which employ standard zero-knowledge techniques, require rewinding in their proof of security and thus are not concurrently secure. An advantage of the simulation paradigm is that our definitions will inherently capture correctness (*i.e.,* if parties honestly follow the protocols then they will each receive their expected outputs). Informally, the security of our system is captured by the following two definitions:

Provider Security: A malicious user (or set of colluding users) must not be able to falsely prove possession of a credential without first obtaining that credential, or arriving at it via an admissible sequence of credential updates under the agreed upon policy. For our purposes, we require that the malicious user(s) cannot provide a proof of being in a state if that state is not present in her credential.

User Security: A malicious provider controlling some collection of corrupted users cannot learn any information about a user's identity or her state in the policy graph beyond what is available through auxiliary information from the environment.

Security for our protocols will be defined using the real-world/ideal-world paradigm, following the approach of [19]. In the real world, a collection of (possibly cheating) users interact directly with a provider according to the protocol, while in the ideal world the parties interact via a trusted party. A protocol is secure if, for every real-world cheating combination of parties we can describe an ideal-world counterpart ("simulator") who gains as much information from the ideal-world interaction as from the real protocol. We note that our definitions will naturally enforce both *privacy* and *correctness*, but not necessarily *fairness*. It is possible that $\mathcal{P}$ will abort the protocol *before* the user has completed updating her credential or proving possession of her credential. This is unfortunately unavoidable in a two-party protocol.

Definition 3.1 (Security for Stateful Anonymous Credentials)**.** Simulation security for stateful anonymous credentials is defined according to the following experiments. Note that we do not explicitly specify auxiliary input to the parties, but this information can be provided in order to achieve sequential composition.

Real experiment. The real-world experiment $\mathbf{Real}_{\hat{\mathcal{P}}, \hat{\mathcal{U}}_1, \ldots, \hat{\mathcal{U}}_\eta}(\eta, k, \Pi_1, \ldots, \Pi_\eta, \Sigma)$ is modeled as k rounds of communication between a possibly cheating provider $\hat{\mathcal{P}}$ and a collection of η possibly cheating users $\{\hat{\mathcal{U}}_1, \ldots, \hat{\mathcal{U}}_\eta\}$. In this experiment, $\hat{\mathcal{P}}$ is given the policy graph for each user $\Pi_1, \ldots, \Pi_\eta$, and the users are given an adaptive strategy Σ that, on input of the user's identity and current graph state, outputs the next action to be taken by the user.

At the beginning of the experiment, all users and the provider conduct the Setup and $\mathsf{ObtainCred}$ procedures. At the end of this step, $\hat{\mathcal{P}}$ outputs an initial state P_1, and each user $\hat{\mathcal{U}}_i$ output state $U_{1,i}$. For each subsequent round $j \in [2, k]$, each user may interact with $\hat{\mathcal{P}}$ to update their credential as required by the strategy Σ. Following each round, $\hat{\mathcal{P}}$ outputs P_j, and the users output $U_{1,j}, \ldots, U_{\eta,j}$, respectively. At the end of the k^{th} round the output of the experiment is $(P_k, U_{1,k}, \ldots, U_{\eta,k})$.

We will define the *honest* provider $\mathcal{P}$ as one that honestly runs its portion of Setup and $\mathsf{ObtainCred}$ in the first round, honestly runs its side of the $\mathsf{UpdateCred}$ and $\mathsf{ProveCred}$ protocols when requested by a user at round $j > 1$, and outputs $P_k = \mathit{params}$ after the final round. Similarly, an honest user $\mathcal{U}_i$ runs the Setup and $\mathsf{ObtainCred}$ protocols honestly in the first round, executes the user's side of the $\mathsf{UpdateCred}$ and $\mathsf{ProveCred}$ protocols, and outputs the credential values received.

Ideal experiment. In the experiment $\mathbf{Ideal}_{\hat{\mathcal{P}}', \hat{\mathcal{U}}'_1, \ldots, \hat{\mathcal{U}}'_\eta}(\eta, k, \Pi_1, \ldots, \Pi_\eta, \Sigma)$ the possibly cheating provider $\hat{\mathcal{P}}'$ first sends the policy graphs $\Pi_1, \ldots, \Pi_\eta$ to the trusted party $\mathcal{T}$. In round $j = 1$, every user $\hat{\mathcal{U}}'_i$ sends a join message to $\mathcal{T}$ of the form (join, i). $\mathcal{T}$ ensures that the policy graph for user i exists, and sets $b_{\mathcal{T}} = 1$ if it exists. $\mathcal{T}$ then forwards $(\mathsf{join}, i, b_{\mathcal{T}})$ to $\hat{\mathcal{P}}'$. $\hat{\mathcal{P}}'$ responds to $\mathcal{T}$ with a bit $b_{\hat{\mathcal{P}}'} \in \{0, 1\}$ that indicates the success or failure of the protocol, and an initial state in Π_i denoted as S_i. If the protocol fails, $S_i = \perp$. Finally, $\mathcal{T}$ returns $(b_{\hat{\mathcal{P}}'} \wedge b_{\mathcal{T}}, S_i)$ to $\hat{\mathcal{U}}'_i$.

In each round $j \in [2, k]$, every user $\hat{\mathcal{U}}'_i$ may send a message to $\mathcal{T}$ of the form $(\mathsf{update}, i, S_i, T_i, t_{id})$ to update her credential using the $\mathsf{UpdateCred}$ protocol with transaction identified by t_{id} to allow for restarts, or (prove, i, S_i) to prove her state using the $\mathsf{ProveCred}$ protocol according to the strategy Σ.

When $\mathcal{T}$ receives an update message, it checks $\hat{\mathcal{U}}'_i$'s current state and policy Π_i to determine whether the requested transition is allowed. If the transition is allowed, or if the transaction id t_{id} was previously seen with the associated transition id, $\mathcal{T}$ sets $b_{\mathcal{T}} = 1$, Otherwise, $b_{\mathcal{T}} = 0$. $\mathcal{T}$ then sends $(\mathsf{update}, t_{id}, b_{\mathcal{T}})$ to $\hat{\mathcal{P}}'$, who responds with a bit $b_{\hat{\mathcal{P}}'} \in \{0, 1\}$ that indicates whether the update should succeed or fail. $\mathcal{T}$ returns $(b_{\hat{\mathcal{P}}'} \wedge b_{\mathcal{T}})$ to $\hat{\mathcal{U}}'_i$.

For a prove message, $\mathcal{T}$ checks that $\hat{\mathcal{U}}'_i$ is in the state in the message (setting $b_{\mathcal{T}}$ to so indicate), and relays $(\mathsf{prove}, b_{\mathcal{T}})$ to $\hat{\mathcal{P}}'$ who responds with a bit $b_{\hat{\mathcal{P}}'}$ to indicate the success or failure of the protocol. $\mathcal{T}$ then sends $(b_{\hat{\mathcal{P}}'} \wedge b_{\mathcal{T}})$ to $\hat{\mathcal{U}}'$.[1] Following each round, $\hat{\mathcal{P}}'$ outputs P_j, and the users output $U_{1,j}, \ldots, U_{\eta,j}$, respectively. At the end of the k^{th} round the output of the experiment is $(P_k, V_j, U_{1,k}, \ldots, U_{\eta,k})$.

We define the *honest* provider $\mathcal{P}'$ as one that sends $\Pi_1, \ldots, \Pi_\eta$ to $\mathcal{T}$, and outputs $b_{\mathcal{P}'} = 1$ and the correct initial state for all users in the first round. In rounds $2, \ldots, k$, $\mathcal{P}'$ returns $b_{\mathcal{P}'} = 1$ for all messages, and outputs $P_k = \epsilon$. Similarly, an honest user $\mathcal{U}'_i$ runs the Setup and $\mathsf{ObtainCred}$ protocol honestly in the first round, makes queries and transitions according to the selection policy in rounds $2, \ldots, k$, and outputs all received credentials in round k.

Let $\ell(\cdot), c(\cdot)$ be polynomially-bounded functions. We now define provider and user security in terms of the experiments above.

Provider Security. A stateful anonymous credential scheme is provider secure if, for every collection of possibly cheating real-world p.p.t. users $\hat{\mathcal{U}}_1, \ldots, \hat{\mathcal{U}}_\eta$, there exists a collection of p.p.t. ideal-world users $\hat{\mathcal{U}}'_1, \ldots, \hat{\mathcal{U}}'_\eta$ such that $\forall \eta = \ell(\kappa)$, $k \in c(\kappa)$, Σ, and every p.p.t. distinguisher:

[1] Note that it is also possible for the user to reveal the state directly to the provider if the credential system is combined with protocols that do not hide the user's actions. The security definition can accommodate for this simply by relaying the state S_i to $\hat{\mathcal{P}}'$

$$\mathbf{Real}_{\mathcal{P},\hat{\mathcal{U}}_1,\ldots,\hat{\mathcal{U}}_\eta}(\eta, k, \Pi_1, \ldots, \Pi_\eta, \Sigma) \overset{c}{\approx} \mathbf{Ideal}_{\mathcal{P}',\hat{\mathcal{U}}'_1,\ldots,\hat{\mathcal{U}}'_\eta}(\eta, k, \Pi_1, \ldots, \Pi_\eta, \Sigma)$$

User Security. A stateful anonymous credential scheme provides user security if, for every real-world p.p.t. provider $\hat{\mathcal{P}}$ who colludes with some collection of corrupted users, there exists a p.p.t. ideal-world provider $\hat{\mathcal{P}}'$ and users $\hat{\mathcal{U}}'$ such that $\forall \eta = \ell(\kappa)$, $k \in c(\kappa)$, Σ, and every p.p.t. distinguisher:

$$\mathbf{Real}_{\hat{\mathcal{P}},\mathcal{U}_1,\ldots,\mathcal{U}_\eta}(\eta, k, \Pi_1, \ldots, \Pi_\eta, \Sigma) \overset{c}{\approx} \mathbf{Ideal}_{\hat{\mathcal{P}}',\mathcal{U}'_1,\ldots,\mathcal{U}'_\eta}(\eta, k, \Pi_1, \ldots, \Pi_\eta, \Sigma)$$

3.2 Basic Construction

In this section, we describe how to realize stateful anonymous credentials. The state records information about the user's attributes, as well as her prior access history. The core innovation here is a protocol for performing state updates, and a technique for "translating" a history-dependent update policy into a cryptographic representation that can be used as an input to this protocol.

The setup, credential granting, and credential update protocols are presented in Figure 1. We will now briefly describe the intuition behind them.

Setup. First, the credential provider $\mathcal{P}$ generates its keypair and identifies one or more access policies it wishes to enforce. Each policy — encoded as a graph — may be applied to one or more users. The provider next "translates" the graph into a cryptographic representation which consists of the graph description and a separate signature for each tag in the graph. The tags embed the graph identity, start state, and end state. The cryptographic policy representations are distributed to users via an authenticated broadcast channel (*e.g.,* by signing and publishing them on a website). The user $\mathcal{U}$ generates a keypair that is certified by the CA.

Obtaining a Credential. When a user $\mathcal{U}$ wishes to obtain a credential, she first negotiates with the provider to select an update policy to which the credential will be bound, as well as the credential's initial state within the policy graph. The user next engages in a protocol to blindly extract a signature under the provider's secret key, which binds the user's public key, her initial state, the policy id, and two random nonces chosen by the user. The *update nonce* N_u is revealed when the user updates the credential and the *usage nonce* N_s is revealed when the user show's her credential. This signature, as well as the nonce and state information, form the credential. While the protocol for obtaining a credential, as currently described, reveals the user's identity through the use of her public key, we can readily apply the techniques found in [14, 15] to provide a randomized pseudonym rather than the public key.

Updating the Credential's State. When the user wishes to update a credential, she first identifies a valid tag within the credential's associated policy representing a transition from her current state to her intended next state, and generates a new pair of nonces. The user creates a commitment embedding these values, as well as her new state from the chosen tag. This commitment and the update nonce from her current credential are sent to the provider to ensure that the credential is not being reused. If the nonce has been previously seen by the provider but associated with a *different* commitment, then the provider knows that the credential is being reused and the provider aborts. If, however, the nonce and commitment are both the same as previously seen, then the user is allowed to continue the protocol. This allows the user to restart previously interrupted updates, albeit in a linkable manner, and the provider is still guaranteed that the user is not creating new credentials since the nonces will remain the same. Of course, if the nonce and commitment have never been seen, then the provider simply adds the pair into a database. The user and provider

$\mathsf{Setup}(\mathcal{U}(1^k), \mathcal{P}(1^k, \Pi_1, \ldots, \Pi_n))$**:** The provider $\mathcal{P}$ generates parameters for the blind signature protocol, as well as for the commitment scheme being used.

Party $\mathcal{P}$ runs KeyGen twice, to create the signature keypairs $(spk_{\mathcal{P}}, ssk_{\mathcal{P}})$ and $(gpk_{\mathcal{P}}, gsk_{\mathcal{P}})$. It retains $(pk_{\mathcal{P}}, sk_{\mathcal{P}}) = ((spk_{\mathcal{P}}, gpk_{\mathcal{P}}), (ssk_{\mathcal{P}}, gsk_{\mathcal{P}}))$ as its keypair. The provider's public key $pk_{\mathcal{P}}$ must be certified by a trusted CA.

Each party $\mathcal{U}$ selects $u \xleftarrow{\$} \mathbb{Z}_q$ and computes the keypair $(pk_{\mathcal{U}}, sk_{\mathcal{U}}) = (g^u, u)$. The user's public key $pk_{\mathcal{U}}$ must be certified by a trusted CA.

For each policy graph Π_i, $\mathcal{P}$ generates a cryptographic representation $\Pi_i^{\mathcal{C}}$.

1. $\mathcal{P}$ parses Π_i to obtain a unique policy identifier pid.
2. For each tag $t = (\mathsf{pid}, S, T)$ in Π_i, $\mathcal{P}$ computes a signature $\sigma_{S \to T} \leftarrow \mathsf{Sign}(gsk_{\mathcal{P}}, (\mathsf{pid}, S, T))$.
3. $\mathcal{P}$ sets $\Pi_i^{\mathcal{C}} \leftarrow \langle \Pi_i, \forall t : \sigma_{S \to T} \rangle$ and publishes this value via an authenticated channel.

$\mathsf{ObtainCred}(\mathcal{U}(pk_{\mathcal{P}}, sk_{\mathcal{U}}, \Pi_i^{\mathcal{C}}), \mathcal{P}(pk_{\mathcal{U}}, sk_{\mathcal{P}}, \Pi_i^{\mathcal{C}}, S))$**:** On input a graph Π_i and initial state S, $\mathcal{U}$ first obtains $\Pi_i^{\mathcal{C}}$. $\mathcal{U}$ and $\mathcal{P}$ then conduct the following protocol:

1. $\mathcal{U}$ picks random usage and update nonces $N_s, N_u \in \mathbb{Z}_q$ and computes $A \leftarrow \mathrm{Commit}(sk_{\mathcal{U}}, N_s, N_u)$.
2. $\mathcal{U}$ conducts an interactive proof to convince $\mathcal{P}$ that A correlates to $pk_{\mathcal{U}}$.
3. $\mathcal{U}$ and $\mathcal{P}$ run the blind signing protocol on committed values so that $\mathcal{U}$ obtains the state signature $\sigma_{\mathsf{state}} \leftarrow \mathsf{Sign}(ssk_{\mathcal{P}}, A)$ with pid, S contributed by $\mathcal{P}$.
4. $\mathcal{U}$ stores the credential $\mathsf{Cred} = (\Pi_i^{\mathcal{C}}, S, \sigma_{\mathsf{state}}, N_s, N_u)$.

$\mathsf{UpdateCred}(\mathcal{U}(pk_{\mathcal{P}}, sk_{\mathcal{U}}, \mathsf{Cred}, T), \mathcal{P}(sk_{\mathcal{P}}, D))$**:** Given a credential Cred currently in state S, $\mathcal{U}$ and $\mathcal{P}$ interact to update the credential to state T:

1. $\mathcal{U}$ parses $\mathsf{Cred} = (\Pi_i^{\mathcal{C}}, S, \sigma_{\mathsf{state}}, N_s, N_u)$ and identifies a signature $\sigma_{S \to T}$ in $\Pi_i^{\mathcal{C}}$ that corresponds to a transition from state S to T (if none exists, $\mathcal{U}$ aborts).
2. $\mathcal{U}$ selects $N_s', N_u' \xleftarrow{\$} \mathbb{Z}_q$ and computes $A \leftarrow \mathrm{Commit}(sk_{\mathcal{U}}, N_s', N_u', \mathsf{pid}, T)$.
3. $\mathcal{U}$ sends (N_u, A) to $\mathcal{P}$. $\mathcal{P}$ looks in the database D for a pair $(N_u, A' \neq A)$. If such a pair is found, $\mathcal{P}$ aborts. Otherwise, $\mathcal{P}$ adds (N_u, A) to D.
4. $\mathcal{U}$ proves to $\mathcal{P}$ knowledge of values $(sk_{\mathcal{U}}, \mathsf{pid}, S, T, N_s', N_u', N_s, \sigma_{\mathsf{state}}, \sigma_{S \to T})$ such that:
   (a) $A = \mathrm{Commit}(sk_{\mathcal{U}}, N_s', N_u', \mathsf{pid}, T)$.
   (b) $\mathsf{Verify}(spk_{\mathcal{P}}, \sigma_{\mathsf{state}}, (sk_{\mathcal{U}}, N_s, N_u, \mathsf{pid}, S)) = 1$.
   (c) $\mathsf{Verify}(gpk_{\mathcal{P}}, \sigma_{S \to T}, (\mathsf{pid}, S, T)) = 1$
5. If these proofs do not verify, $\mathcal{P}$ aborts. Otherwise $\mathcal{U}$ and $\mathcal{P}$ run the blind signing protocol on committed values to provide $\mathcal{U}$ with $\sigma_{\mathsf{state}}' \leftarrow \mathsf{Sign}(ssk_{\mathcal{P}}, A)$.
6. $\mathcal{U}$ stores the updated credential $\mathsf{Cred}' = (\Pi_i^{\mathcal{C}}, T, \sigma_{\mathsf{state}}', N_s', N_u')$.

Figure 1: Basic protocols for obtaining and updating a stateful anonymous credential.

then interact to conduct a zero-knowledge proof that: (1) the remainder of the information in the commitment is identical to the current credential, (2) the user has knowledge of the secret key corresponding to her current credential, and (3) the cryptographic representation of the policy graph contains a signature on a tag from the current state to the new state. If these conditions are met, the user obtains a new credential embedding the new state.

Showing a Credential. In Figure 2, we provide the protocol for anonymously proving possession of a single-show stateful credential. This protocol runs in one of two modes. If the stateful credential system is being used to control access to typical, non-oblivious services, such as digital rights management for an online music service, then the user can reveal her state directly to the provider. When the stateful credential system is combined with an oblivious protocol where the user's actions

$\mathsf{ProveCred}(\mathcal{U}(pk_{\mathcal{P}}, sk_{\mathcal{U}}, \mathsf{Cred}), \mathcal{P}(pk_{\mathcal{P}}, E))$**:** User $\mathcal{U}$ proves knowledge of the Cred as:

1. $\mathcal{U}$ parses Cred as $(\Pi_i^{\mathcal{C}}, S, \sigma_{\mathsf{state}}, N_s, N_u)$. and sends its usage nonce N_s to $\mathcal{P}$ (who aborts if $N_s \in E$).
2. If $N_s \in E$, $\mathcal{P}$ aborts. Otherwise, $\mathcal{U}$ continues by sending either:
   (a) The current credential state S to $\mathcal{P}$ in the clear when the user's actions are not hidden.
   (b) A commitment on the current credential state $A \leftarrow \text{Commit}(S)$ to $\mathcal{P}$ when the user's actions are hidden.
3. $\mathcal{U}$ then conducts an interactive proof to convince $\mathcal{P}$ that it possesses a signature σ_{state} embedding N_s, S, and that it has knowledge of the secret key $sk_{\mathcal{U}}$.
4. $\mathcal{P}$ adds N_s to E.

Figure 2: Basic protocol for proving knowledge of a single-show anonymous credential.

must be hidden, however, the user instead embeds her state into a commitment. The user then proves knowledge of: (1) a signature containing the state value and usage nonce from her current credential signed by the provider, and (2) a secret key that is also contained within the signature.

Theorem 3.2. *The stateful anonymous credential system presented in Figures 1 and 2 are user and provider secure under the Strong RSA (respectively, LRSW, Strong Diffie-Hellman) assumption when instantiated with the RSA-based Camensich-Lysyanskaya (respectively, bilinear Camenisch-Lysyanskaya, Boneh-Boyen) signature scheme.*

Proof sketch. We consider the user and provider security separately. We begin by describing the operation of a p.p.t. simulator for the ideal-world instantiations of the protocol, and then argue the computational indistinguishability between $\mathbf{Real}_{\hat{\mathcal{P}},\hat{\mathcal{U}}_1,\ldots,\hat{\mathcal{U}}_\eta}(\eta, k, \Pi_1, \ldots, \Pi_\eta, \Sigma)$ and $\mathbf{Ideal}_{\hat{\mathcal{P}}',\hat{\mathcal{U}}'_1,\ldots,\hat{\mathcal{U}}'_\eta}(\eta, k, \Pi_1, \ldots, \Pi_\eta, \Sigma)$ via a hybrid argument. Specifically, we define distributions **Game 0**$, \ldots,$ **Game 4**, with **Game 0** $=$ $\mathbf{Real}_{\hat{\mathcal{P}},\hat{\mathcal{U}}_1,\ldots,\hat{\mathcal{U}}_\eta}$ and **Game 4** $=$ $\mathbf{Ideal}_{\hat{\mathcal{P}}',\hat{\mathcal{U}}'_1,\ldots,\hat{\mathcal{U}}'_\eta}$. Intermediary distributions **Game 1**$, \ldots,$ **Game 3** examine changes made to the commitment, zero-knowledge proof, and blind signature components of our protocols in our simulator. To prove indistinguishability, we examine the statistical difference between successive distributions to bound the advantage of a p.p.t. distinguisher D. We denote the advantage of D in distinguishing the output of **Game i** as:

$$Pr[\mathbf{Game\ i}] = Pr[D(X) = 1 : X \xleftarrow{\$} \mathbf{Game\ i}]$$

User Security: To prove user security, we first describe a simulator for the ideal-world provider $\hat{\mathcal{P}}'$ with rewindable black box access to a real-world provider $\hat{\mathcal{P}}$. Upon initialization, $\hat{\mathcal{P}}$ outputs the system parameters, and the cryptographic representation of the policy graphs for each user $\Pi_1^{\mathcal{C}}, \ldots, \Pi_\eta^{\mathcal{C}}$. The simulator $\hat{\mathcal{P}}'$ checks that the parameters and the signatures on the graphs are valid, and if they are sends the plaintext policy graphs $\Pi_1, \ldots, \Pi_\eta$ to the trusted part $\mathcal{T}$. Otherwise, $\hat{\mathcal{P}}'$ aborts and sends $\Pi_i = \bot$ to $\mathcal{T}$.

During round 1 of the simulation, $\hat{\mathcal{P}}'$ may receive a *join* message of the form $(join, i, b_{\mathcal{T}})$ from each of the η users. When $\hat{\mathcal{P}}'$ receives such a message, it generates a user keypair according to the procedure outlined in Setup for the user $\mathcal{U}_i$. $\hat{\mathcal{P}}'$ then runs the $\mathsf{ObtainCred}$ protocol with $\hat{\mathcal{P}}$ using $\mathcal{U}_i$'s policy graph and locally generated keypair. If at any point the protocol fails, $\hat{\mathcal{P}}'$ aborts and sends $b_{\hat{\mathcal{P}}'} = 0$ and $S_i = \bot$ to $\mathcal{T}$. If the protocol succeeds, $\hat{\mathcal{P}}'$ checks the received state signature and sends the user's state S_i and $b_{\hat{\mathcal{P}}'} = 1$ to $\mathcal{T}$. In addition, $\hat{\mathcal{P}}'$ also stores the user's information, including the keypair that was generated, her policy graph, her start state, and the state signature returned after completion of the ObtainCred protocol. At the end of round 1, $\hat{\mathcal{P}}'$ will have a local copy of the current credentials for every user under the keys that were generated when the *join* message was received.

When $\hat{\mathcal{P}}'$ receives a $(update, t_{id}, b_{\mathcal{T}})$ message in rounds $2, \ldots, k$, it first checks to see if the value t_{id} has been seen previously. If it has, then $\hat{\mathcal{P}}'$ chooses a user at random from among those it has already used in an UpdateCred protocol, and reuses the same commitment and nonce that was previously used. Otherwise, $\hat{\mathcal{P}}'$ selects a user uniformly at random from among all credentials it has stored, and chooses a valid transition from that user's current state (as represented in the locally stored credential) to its next state. Such a transition is guaranteed to exist for all users due to our use of null transitions on terminal states. Next, $\hat{\mathcal{P}}'$ runs the UpdateCred protocol with $\hat{\mathcal{P}}$, acting as the selected user with either a reused commitment and nonce pair, or a valid transition. If at any point the UpdateCred protocol fails, $\hat{\mathcal{P}}'$ returns $b_{\hat{\mathcal{P}}'} = 0$ to $\mathcal{T}$. If the protocol succeeds, $\hat{\mathcal{P}}'$ checks that the new state signature is valid and sends $b_{\hat{\mathcal{P}}'} = 1$ to $\mathcal{T}$. $\hat{\mathcal{P}}'$ also updates the credential for the user that was selected to reflect the change in state.

Finally, when $\hat{\mathcal{P}}'$ receives a $(prove, b_{\mathcal{T}}$ message in rounds $2, \ldots, k$, it chooses a random user from among those credentials it has stored and runs ProveCred with $\hat{\mathcal{P}}$ as the selected user. If the protocol fails at any time, $b_{\hat{\mathcal{P}}'} = 0$ is sent to $\mathcal{T}$, and otherwise $b_{\hat{\mathcal{P}}'} = 1$ is sent.

Now, we examine the statistical difference between successive distributions. In **Game 0** = $\mathbf{Real}_{\hat{\mathcal{P}}, \mathcal{U}_1, \ldots, \mathcal{U}_\eta}$, the cheating real-world provider $\hat{\mathcal{P}}$ interacts with honest users $\mathcal{U}_1, \ldots, \mathcal{U}_\eta$. By the notation described above, we have

$$Pr[\mathbf{Game\ 0}] = Pr[D(X) : X \xleftarrow{\$} \mathbf{Real}_{\hat{\mathcal{P}}, \mathcal{U}_1, \ldots, \mathcal{U}_\eta}]$$

Game 1 operates exactly as **Game 0**, except here we replace the commitments used in the real-world protocols with honest users with our simulated commitments of randomly chosen user credentials maintained by $\hat{\mathcal{P}}'$. It suffices to show that by the hiding property of the commitment scheme, the commitment values of our randomly chosen users is distributed indistinguishably from those of the honest users.[2] Thus, for some negligible function $\nu_1(\kappa)$ we have $Pr[\mathbf{Game\ 1}] - Pr[\mathbf{Game\ 0}] \leq \nu_1(\kappa)$.

In **Game 2** we operate as in **Game 1**, but replace the zero-knowledge proofs of the credential values provided by honest users with those of the credentials maintained by $\hat{\mathcal{P}}'$ in our simulator. Again, the crux of this argument lies in showing that the zero-knowledge property of the proofs ensures that the two distributions are indistinguishable. By the witness-indistinguishability property of zero-knowledge proofs of knowledge, we have $Pr[\mathbf{Game\ 2}] - Pr[\mathbf{Game\ 1}] < \nu_2(\kappa)$ for a negligible function $\nu_2(\kappa)$.[3]

We define **Game 3** to be identical to **Game 2**, but now we replace the state signatures produced by the blind signature scheme with an honest user with the signatures obtained by our simulator on the simulated user credentials. As with the previous components, the signatures produced by the blind signature must be uniformly distributed in the underlying group. However, we must also require that the interaction in the blind signatures scheme itself does not leak information about the contents of the message being signed. To do so, we can instantiate our signature scheme with the RSA-based Camenisch-Lysyanskaya [15], bilinear Camenisch-Lysyanskaya [16], or Boneh-Boyen [5]

[2] When using Pedersen commitments [46], the commitment values are perfectly hiding, and when we use Fujisaki-Okamoto commitments [33] the values are statistically hiding.

[3] Many perfect zero-knowledge proof protocols exist, such as [26], which imply the witness indistinguishability property with no additional computational complexity assumptions.

blind signature schemes, and apply the transformations described by Fischlin and Schröder [32] to make them selective-failure blind signatures. These signatures are secure under the Strong RSA, LRSW, and Strong Diffie-Hellman assumptions, respectively. Therefore, we have $Pr[\textbf{Game 3}] - Pr[\textbf{Game 2}] \leq \nu_3(\kappa)$.

Finally, we define **Game 4** to be the interaction of our simulated ideal-world cheating provider $\hat{\mathcal{P}}'$ with the trusted party $\mathcal{T}$. This interaction incorporates all of the changes outlined above, and we can bound D's advantage in distinguishing **Game 0** from **Game 4** by summing the statistical difference:

$$Pr[D(X) = 1 : X \xleftarrow{\$} \textbf{Ideal}_{\hat{\mathcal{P}}', \mathcal{U}'_1, \ldots, \mathcal{U}'_\eta}] - Pr[D(X) = 1 : X \xleftarrow{\$} \textbf{Real}_{\hat{\mathcal{P}}, \mathcal{U}_1, \ldots, \mathcal{U}_\eta}] \leq \nu_1(\kappa) + \nu_2(\kappa) + \nu_3(\kappa)$$

Provider Security: To prove provider security, we first describe simulators for the ideal-world users $\hat{\mathcal{U}}'_1, \ldots, \hat{\mathcal{U}}'_\eta$ with rewindable black box access to the real-world users $\hat{\mathcal{U}}_1, \ldots, \hat{\mathcal{U}}_\eta$ using strategy Σ. For clarity, we refer to the simulator as $\hat{\mathcal{U}}'_i$ with access to $\hat{\mathcal{U}}_i$. Upon initialization, $\hat{\mathcal{U}}'_i$ generates the provider keypairs according to the Setup protocol and signs each of the policy graphs to create their cryptographic representations $\Pi_1^{\mathcal{C}}, \ldots, \Pi_\eta^{\mathcal{C}}$. $\hat{\mathcal{U}}'_i$ sends the parameters and policy graphs to $\hat{\mathcal{U}}$.

In round 1, $\hat{\mathcal{U}}_i$ will initiate the ObtainCred protocol with $\hat{\mathcal{U}}'_i$. When the protocol is initiated, $\hat{\mathcal{U}}'_i$ will send a $(join, i)$ message to the trusted party $\mathcal{T}$. If $\mathcal{T}$ responds with a bit $b = 1$ and state S_i, $\hat{\mathcal{U}}'_i$ runs the protocol as usual and embeds the returned state S_i into the user's state signature. Otherwise, $\hat{\mathcal{U}}'_i$ aborts the protocol.

When $\hat{\mathcal{U}}_i$ initiates the UpdateCred protocol in rounds $2, \ldots, k$, $\hat{\mathcal{U}}'_i$ runs the protocol as usual by checking for a reused update nonce, and verifying the user's proof of knowledge on her credential state. If the nonce was previously used and the commitment provided remains the same, $\hat{\mathcal{U}}'_i$ sets t_{id} to be the same as that which was used in the transaction with the repeated nonce, commitment pair. If the nonce is new, t_{id} is chosen randomly by $\hat{\mathcal{U}}'_i$. In addition, $\hat{\mathcal{U}}'_i$ rewinds $\hat{\mathcal{U}}_i$ and uses the extractor for the proof of knowledge to reveal the user's identity i, current state S_i, and intended next state T_i. Then, $\hat{\mathcal{U}}'_i$ sends the message $(update, i, S_i, T_i, t_{id})$ to $\mathcal{T}$. If $\mathcal{T}$ returns $b = 1$, the protocol completes as usual and $\hat{\mathcal{U}}'_i$ sends a signature on the new state to $\hat{\mathcal{U}}_i$, and otherwise the protocol is aborted.

Finally, if the user $\hat{\mathcal{U}}_i$ initiates the ProveCred protocol in rounds $2, \ldots, k$, $\hat{\mathcal{U}}'_i$ runs the protocol by checking for reuse of the usage nonce provided by $\hat{\mathcal{U}}_i$ and verifying the user's proof of her current state. $\hat{\mathcal{U}}'_i$ then rewinds $\hat{\mathcal{U}}_i$ and uses the extractor for the proof of knowledge to reveal the user's identity and current state S_i. $\hat{\mathcal{U}}'_i$ then sends $(prove, i, S_i)$ to $\mathcal{T}$, and completes the protocol if $b = 1$ is returned or aborts if $b = 0$ is returned.

We now investigate the differences between successive distributions. We define $\textbf{Game 0} = \textbf{Real}_{\mathcal{P}, \hat{\mathcal{U}}_1, \ldots, \hat{\mathcal{U}}_\eta}$ as the distribution where the honest real-world provider $\mathcal{P}$ interacts with the potentially cheating users $\hat{\mathcal{U}}_1, \ldots, \hat{\mathcal{U}}_\eta$. By the notation described above, we have

$$Pr[\textbf{Game 0}] = Pr[D(X) : X \xleftarrow{\$} \textbf{Real}_{\mathcal{P}, \hat{\mathcal{U}}_1, \ldots, \hat{\mathcal{U}}_\eta}]$$

Game 1 is defined exactly as **Game 0**, except that we consider the ability of our extractor to correctly reveal the values in the zero-knowledge proofs in the UpdateCred and ProveCred protocols. If the extractor cannot extract the value, the simulator would be unable to pass the correct state information to $\mathcal{T}$ for further processing. The probability of failure for our extractor is bounded by the knowledge error of the proof of knowledge, which we define as $\nu_1(\kappa)$.[4]. Since we must run the

[4] For the Schnorr technique in a group of order q, this error is $1/q$

extractor once for every message (either *update* or *prove*) received from each user during rounds $j = 2, \ldots, k$, the statistical difference is bounded by:

$$Pr[\textbf{Game 1}] - Pr[\textbf{Game 0}] \leq k\eta\nu_1(\kappa)$$

In **Game 2** we operate as in **Game 1**, but now consider the ability of the user to alter the value of the commitment between the proof of knowledge discussed in **Game 1** and the blind signature protocol in the UpdateCred protocol. If, for example, the user were able to prove knowledge of the correct state and then alter the commitment to receive a blind signature on a different state, the future rounds of the protocol would produce errors when validating the user's state with $\mathcal{T}$. The binding property of the commitment scheme prevents such tampering with negligible probability $\nu_3(\kappa)$.[5] In this case, $Pr[\textbf{Game 2}] - Pr[\textbf{Game 1}] \leq k\eta\nu_2(\kappa)$.

We define **Game 3** to be identical to **Game 2**, though we now consider the ability of the user to forge a valid blind signature on her state information. To do so, the user would have to violate the existential unforgeability property of the blind signature scheme used. For the RSA-based Camenisch-Lysyanskaya [15], bilinear Camenisch-Lysyanskaya [16], and Boneh-Boyen [5] blind signature schemes, existential unforgeability is proven under the Strong RSA, LRSW, and Strong Diffiie-Hellman assumptions, respectively. Thus, $Pr[\textbf{Game 3}] - Pr[\textbf{Game 2}] \leq k\eta\nu_3(\kappa)$.

Finally, we define **Game 4** to be the interaction of our simulated ideal-world cheating users $\hat{\mathcal{U}}'_1, \ldots, \hat{\mathcal{U}}'_\eta$ with the trusted party $\mathcal{T}$. This interaction incorporates all of the changes outlined above, and we bound D's advantage in distinguishing **Game 0** from **Game 4** by summing the statistical difference:

$$Pr[D(X) = 1 : X \overset{\$}{\leftarrow} \textbf{Ideal}_{\hat{\mathcal{P}}', \mathcal{U}'_1, \ldots, \mathcal{U}'_\eta}] - Pr[D(X) = 1 : X \overset{\$}{\leftarrow} \textbf{Real}_{\hat{\mathcal{P}}, \mathcal{U}_1, \ldots, \mathcal{U}_\eta}] \leq \nu_1(\kappa) + k\eta(\nu_2(\kappa) + \nu_3(\kappa))$$

□

3.3 Multi-show Credential Extension

The stateful anonymous credential system as we have described it thus far enables the user to prove possession and update a given credential at most once. Specifically, the usage and update nonces that are revealed during their respective protocols gives the provider a means of identifying reused credentials. In order to adapt this system to a k-times anonymous credential system, we can use pseudorandom functions to replace the nonces, as shown in [11]. To do so, the user can keeps a counter $0 < i \leq k$ and reveals the output of a pseudorandom function evaluated on the counter $f_s(i)$. For our purposes, the seed s can simply be set to be the nonces described in our original protocols.

In order to ensure that the credential is limited to at most k uses, the user proves (in zero-knowledge) that seed of the function is the same as the random nonce contained in her credential and that the counter used as input lies in the known range $[1, k]$. This proof would be performed in conjunction with those already described in the respective protocols, and can be performed with standard zero-knowledge techniques [21, 17, 17, 6]. Since the output of the pseudorandom function is indistinguishable from a random element of the group, the user is assured that its use does not leak information about her identity, the nonce used as the seed, or the counter value beyond what is proved in the zero-knowledge proof. The pseudorandom function above can be instantiated with

[5] This property holds under the discrete logarithm assumption for Pedersen commitments [46] and the factoring assumption for Fujisaki-Okamoto commitments [33].

either the Dodis-Yampolskiy [31] or Naor-Reingold [44] pseudorandom functions, which achieve security under the y-Decisional Diffie-Hellman Inversion and Decisional Diffie-Hellman assumptions, respectively.

It is also important to note that the use of these pseudorandom functions can impact our stateful credential system in multiple ways. When the pseudorandom function is used to replace the usage nonce N_s in the ProveCred protocol, our stateful credential system becomes a k-times anonymous credential system like any other. However, when used to replace the update nonce N_u, the pseudorandom function enables the user to "clone" her current state up to k-times. This, in turn, enables our stateful credential system to model k concurrent user behaviors, which may be of value in certain scenarios, such as parallel execution of programs. Moreover, it is even possible to attach different values of k to the tags in the graph. In this case, tags of the form $(\mathsf{pid}, S \rightarrow T, k_{S \rightarrow T})$ are created, where $k_{S \rightarrow T}$ indicates the number of times the user will be allowed to reuse the credential when they use the tag associate with edge $S \rightarrow T$. When the user creates the commitment for her new credential in the UpdateCred protocol, she embeds the value k_{StoT}, proves that this value is the same as the one embedded in the tag, and then completes the blind signature protocol with the provider using this commitment. By attaching different values of k to each tag, we are able limit the update and usage behavior of the resultant credential according to the user's current state and her access history.

3.4 Auditing Extensions

One natural extension for our credential system is to augment it with an auditing mechanism that records the actions of users as they transition through their associated policy graph. Such audit logging is important in many real-world applications where the provider needs to keep track of potentially malicious user behavior, or mine that information for business purposes. We note that audit logging is in natural opposition to the notion of user security that we have provided in our definitions. While developing weaker security models is an interesting area of future work, we instead focus on a narrow set of audit methods that require only limited extensions to our existing model. Specifically, we consider scenarios where (1) there exists a trusted party who is granted the ability to selectively revoke anonymity, and (2) a model where the user's privacy is guaranteed only until she misbehaves by attempting to reuse a credential.

Trusted Party Auditing. In this scenario, we extend our model to include a trusted third party who will maintain encrypted versions of each users' actions within the credential system and can reveal these actions to the provider when presented with some evidence of wrongdoing. To begin, the trusted party generates and publishes a keypair. During the ProveCred and UpdateCred protocols, the user creates a verifiable encryption of the current contents of her credential under the trusted party's public key, which may include her identity, her current state, and the nonces she is using, and sends the encryption to the provider. In addition, the user provides a (non-interactive) zero-knowledge proof that the contents of the encryption are the same as those of her current credential and that it has been encrypted under the trusted party's public key. If the proof verifies, the provider will store this information, along with information about the current transaction, such as the nonce and commitment values used. When the provider detects malicious behavior, it can send the information about that transaction to the trusted party who may, if there is acceptable cause, decrypt the associated encrypted credentials. This information allows the provider to learn the user's identity, and the state of the credential being reused. If the credential is a k-time use credential, as described above, the provider can also use the pseudorandom function seed values contained in the encrypted credential information to reproduce the outputs for the pseudorandom function for the reused credential, thereby learning the other transactions performed by the given user with that credential.

Double Spend Auditing. To remove the trusted third party from our auditing extension, we can make use of the double spending tracing method first developed by Camenisch, Hohenberger, and Lysyanskaya [12]. In this auditing scheme, the provider is given verifiable encryptions of the users' actions, and the secret key for these encryptions is revealed when the user attempts to reuse a credential. Specifically, the user generates a keypair for a verifiable encryption algorithm, such as bilinear El Gamal encryption, and publishes the public key. Again, the user creates a verifiable encryption of her credential contents under her verifiable encryption secret key during the ProveCred and UpdateCred protocols, and sends that encryption to the provider. In addition, the user sends a token of the form $T = sk_{\mathcal{U}} f_t(i)^R$, where $f_t(\cdot)$ is a pseudorandom function with seed t, i is a counter for her uses of the current credential, R is a randomly chosen value selected by the provider, and $sk_{\mathcal{U}}$ is her secret verifiable encryption key. A zero-knowledge proof ensures that the values in the encryption are consistent with her current credential, that it has been encrypted under the user's secret verifiable encryption key, and that key is embedded in the token T. Notice that if the user wishes to reuse a credential, then she must provide two tokens T_1 and T_2, where every value is the same except for the random values R_1 and R_2, respectively. In that case, the user's secret key $sk_{\mathcal{U}}$ can be recovered through the following equation:

$$\left(\frac{T_2^{R_1}}{T_1^{R_2}}\right)^{(R_1-R_2)^{-1}}$$

Once the provider has retrieved the secret key, the verifiable encryption can be decrypted and the provider will then have access to the malicious user's credential information, as described above.

3.5 Efficiency

Since our stateful anonymous credential system is intended the behavior of users in real-time, the efficiency of our protocols is of the utmost importance. Here, we explore the time and space complexity for the user and provider. For the purposes of our analysis, we consider the complexity of our system with respect to the number of tags (i.e., edges) in a single policy graph. We denote the number of tags in policy graph i as n_i.

For users of the stateful credential system, the space complexity of the protocols is contingent upon the size of the credential that they hold. With our basic system shown in Figures 1 and 2, a credential contains the cryptographic representation of the policy graph, nonces, the user's current state, and a signature on those values. Therefore, the credential must take up space that is $O(n_i)$ for the user's policy graph i due to the cryptographic representation of the policy containing each tag in the graph. Of course, if the user had access to a reliable policy graph server from which to she could anonymously retrieve tags, then the user would only have to maintain her current state, nonces, and a signature on these items. With regard to the time complexity, each of the protocols in our credential system requires only a constant number of steps for the users. The most time consuming of these is the proof of knowledge used in the ProveCred and UpdateCred protocols, however even these proofs only examine a constant number of values contained within the user's credential. Thus, the overall time complexity for the user is $O(1)$ throughout the stateful credential system.

The provider in our system must maintain the policy graphs for all policies that are active within the system, as well as a database containing the nonces used in the ProveCred and UpdateCred protocols. The space complexity of the policy graphs is clearly $O(\sum_{\forall i} n_i)$, however, it is more difficult to quantify the space complexity of the nonce databases since they are contingent upon the paths that the users take in their associated policy graphs. If these graphs contain cycles, then the only guarantee is that the databases must be polynomial in size since the a poly-time user is naturally limited in the number of transitions they can make. The provider's time complexity differs within each protocol in our system. During the Setup protocol, the provider must sign each tag in all graphs, thereby making the time complexity of that protocol $O(\sum_{\forall i} n_i)$. For the ObtainCred and UpdateCred protocols, the provider participates in an interactive zero-knowledge proof of the values in the user's credential, which are constant in the size of the graph, and performs the blind signature protocol to output the user's newest state signature. Assuming that the time complexity of the blind signature protocol being used is linear in the number of items being signed, then the overall complexity for the provider in these protocols is still $O(1)$ in the size of the user's policy graph. Finally, in the ProveCred protocol, the provider need only participate in a single interactive zero-knowledge proof of the user's credential state, and so the complexity is also $O(1)$.

Practically speaking, the complexity of our stateful credential system imposes few, if any, hardships on the user since their time complexity is constant, and at worst their space complexity is linear in the size of their associated policy graph. For the provider, however, there are several potential pitfalls in the use of this system in practical applications. For one, the Setup protocol requires the provider to sign every tag in every graph used within the system. Luckily, this signing procedure need only be run once at the initialization of the system, and so its time complexity can be amortized over the entire life of the system. Moreover, when the provider needs to make changes to the policy graphs, she only needs to re-sign those tags (i.e., edges) that have been altered. Another potential issue arises when considering the size of the graphs and the type of user behaviors that they can encode. As mentioned earlier, we can assume that a user is only able to make a polynomial number of state transitions in the system, but these transitions may represent the user's choice from among several disjoint paths. In fact, encoding this type of combinatorial behavior would require graphs that are superpolynomial in size since that graph would have to anticipate all possible user actions, thereby limiting the stateful credential system to representing a limited depth of history.

4 Controlling Access to Anonymous and Oblivious Protocols

Thus far, we have presented a stateful anonymous credential system that efficiently models the user's actions while simultaneously hiding her identity and current state. In this section, we show how to augment this credential system to implement real-world access control policies within a variety of anonymous and oblivious cryptographic systems. These access control policies can restrict the user's actions according to not just their identity, but also her current state and past history of actions. At a high level, the access control mechanism works as follows. We first extend the tags in our policy graphs to include the action that the user is allowed to perform when they make the associated transition in the policy graph. These new tags are of the form $(\mathsf{pid}, S \to T, j)$, where pid is the policy graph identity, $S \to T$ is the edge associated with the tag, and j is the identity of the action or item that the user is allowed access to. Furthermore, each edge in the policy graph can be associated with many tags, since there is now one tag per allowable action. Also, as described in Section 3, null transactions are placed on all terminal states, which allow the user to perform a predefined null action. The specifics of action identity are based on the cryptographic system that we are combining our credential system with, such item indices for oblivious transfer or hashed keywords in oblivious keyword search. Figure 3 shows an example access graph.

When the user wishes to perform action j, she first finds a tag that includes the outgoing

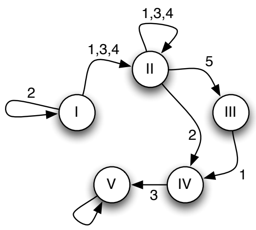

Figure 3: Sample access policy for a small oblivious database. The labels on each transition correspond to the database item indices that can be requested when a user traverses the edge, with null transitions represented by unlabeled edges.

edge from her current state and the desired action. The user then runs the AccessAndUpdate protocol with the provider, which combines the UpdateCred and ProveCred protocols from our original stateful credential system. This combined protocol first requires the user to prover her credential state and the existence of a tag that includes that state, as well as the requested action for the associated cryptographic system being protected. Upon successful completion of the proof, the user's requested action is performed within the associated cryptographic system, and at the conclusion of this action the user receives a new state signature that indicates the transition from her current state to the destination state within the tag used. In combining the two protocols, we ensure that the user must be in a valid state to perform her requested action and, once that action is complete, the user's state is updated to reflect that action. At the same time, the user's identity and action remain hidden from the provider, and the provider is guaranteed that the user is following a valid access control policy graph.

4.1 Access Control Policies from Stateful Credentials

One benefit of our state machine-based policy graphs is that we can use them to implement a wide variety of real-world access control models. In general, we can implement any access controls based on the state transition model, including the Biba [3], Clark-Wilson [25], Bell-La Padula [2], and Brewer-Nash [8] access control models. Here, we describe how to construct policy graphs for popular access control models, and discuss their associated efficiency considerations. Furthermore, we provide two extensions that limit the size of the policy graphs by grouping actions into classes.

Discretionary Access Controls. One of the most widely used forms of access controls are discretionary access controls [30], where access permissions are applied arbitrarily by the computer operator or systems administrator. This model is found on commercial operating systems, including Apple's OSX and Microsoft's Windows. Generally, discretionary access controls can be implemented in one of two ways: (i) access control lists, or (ii) capability lists [40]. Access control lists, or ACLs, are defined with respect to individual objects, and define the users that are allowed access to those objects. Capability lists, on the other hand, are defined with respect to users, and define the objects that the user is allowed to access. Since our stateful credential system is inherently user-centric, implementation of a capability list for each user is fairly straightforward. To do so, we create a policy graph for each of the p users, or groups thereof, containing a single state with a self-loop. Tags associated with the self-loop are created for each object that the user can access, thereby creating $O(np)$ tags within the system, where n is the number of objects and p is the number of users in the system.

Brewer-Nash Model. The Brewer-Nash model [8], otherwise known as the Chinese Wall, is a mandatory access control model that is widely used in the financial services industry to prevent

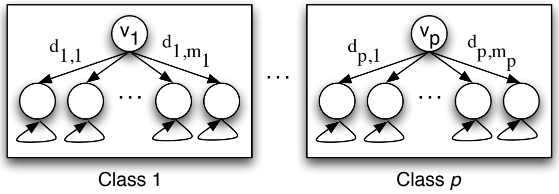

Figure 4: Example access graphs for the Brewer-Nash model. The user receives one access graph per class, where each graph allows access to at most one of the datasets $d_{i,j}$ for the associated conflict of interest class i.

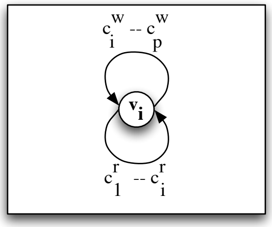

Figure 5: Example access graph for a user with security level i in the Bell-LaPadula model. The graph allows read access to all resources in classes c_1^r through c_i^r and write access to all objects in classes c_i^w to c_p^w.

an employee from working on the finances of two companies that are in competition with one another. Intuitively, the resources in the system are first divided into groups called *datasets*. These datasets are further grouped into *conflict of interest classes* such that all of the companies that are in competition with one another have their datasets in the same class. The model ensures that once a user chooses an object from a dataset in a given class, that user has unrestricted access to all objects in the selected dataset, but no access to objects in any other dataset in that class.

In Figure 4, we show an implementation of the Brewer-Nash model in our stateful credential system. The implementation for this access control model consists of disjoint policy graphs for each of the p conflict of interest classes. Each graph Π_i, has a single start state and m_i additional states that represent the datasets, denoted as $d_{i,j}$, within the associated conflict of interest class. The tags for the edges from the start state to the other states in the graph force the user to select only one of the objects within the allowed datasets in that conflict of interest class, and the self-loops on the destination states ensure that the user has continued access to the objects that make up her chosen dataset, but no access to other datasets within the class. For each edge, the number of tags created is linear in the number of objects within the associated dataset, and therefore the overall complexity of this access control model implementation is $O(n)$. The user initially obtains separate credentials for each of the p access policy graphs and is allowed to move within them independently, thereby allowing the user access to exactly one dataset (and its respective objects) within each conflict of interest class.

Bell-La Padula Model. Another well-known mandatory access control model is the Bell-La Padula model [2], also known as the Multilevel Security model. The Bell-La Padula model is designed with the intent of maintaining data confidentiality in a classified computer system, and it is typically used in high security environments. In this security model, resources and users are labeled with a security level (*e.g.,* top secret, secret, etc.). The security level labels follow a partial ordering and provide a hierarchy that describes the sensitivity of information with higher level classes indicating more sensitive information. The two basic properties of the Bell-LaPadula model state that a user cannot read a resource with a security level greater than her own, and she cannot write to resources with a security level less than her own. Therefore, the model ensures that information from highly sensitive objects cannot be written to low security objects by using the user as an intermediary.

In Figure 5, we illustrate the Bell-La Padula model as implemented in our stateful credential system. For each security level $i = 1, \ldots, p$, we create a policy graph with a single state and a self-loop. The tags associated with this self-loop allow read access to objects whose security level is less than or equal to the graph's security level, denoted as $c_1^r, \ldots, c_i^r$. Additionally, tags are also created to allow write access to objects with security level greater than or equal to the current security level, denoted as $c_i^w, \ldots, c_p^w$. To accommodate for both read and write access, the object identities in the system can simply be split into two ranges for each type of access. The complexity of this model is therefore $O(np)$, since each of the p access policy graphs must contain all n unique items, either as a read or write access.

Extensions for Compact Access Graphs. Thus far, the protocol requires that a tag in the policy graph must be defined on every object in the cryptographic system being protected. Yet, there are cases where many objects may have the same access rules applied, and therefore we can reduce the number tags used by referring to the entire group with a single tag. A simple solution to group objects into classes is to replace specific object identities with general equivalence class identities in the graph tags. Specifically, if the number of object identities is sufficiently small (i.e., $\log_2(q)/2$ for the group $\mathbb{Z}_q$), then we can relabel an object j with values of the form $(c||j) \in \mathbb{Z}_q$, where c is the identity of the item class, and $||$ represents concatenation. During the AccessAndUpdate protocol, the user can obtain any object $(c||j)$ by performing a zero-knowledge proof on the first half of the label, showing that the selected tag contains the class c.

An alternative approach requires the provider to arrange the identities of objects in the same class so that they fall into contiguous ranges. We replace the object identities in the tags with *ranges* of objects to create tags of the form $(\mathsf{pid}, S \rightarrow T, j, k)$, which allows access to any object with an identity in $[j, k]$. We slightly change the AccessAndUpdate protocol so that rather than proving equality between the requested object and the object present in the tag, the user now proves that the requested object lies in the range embedded in the user selected tag, as described by the hidden range proof technique in Section 2. Notice that while this approach requires that the database be reorganized such that classes of items remain in contiguous index ranges, it can be used to represent more advanced data structures than the class label technique described above, such as hierarchical classes.

5 Access Controls for an Oblivious Transfer Database

Oblivious transfer and private information retrieval protocols are ideal building blocks for constructing privacy-preserving databases that allow users to retrieve records without revealing their identity or their choice of database records. This is particularly useful in highly-sensitive applications where even the history of database accesses could reveal sensitive information about the user or the records themselves. In this section, we show how we can augment adaptive oblivious transfer (OT) protocols with our stateful anonymous credential system to provide efficient access controls for oblivious databases. Specifically, we show how to couple stateful credentials with the recent standard-model adaptive OT scheme due to Camenisch, Neven and shelat [19]. Similar methods can be used to couple our system with other oblivious transfer schemes, such as those of Green and Hohenberger [38], in addition to many other oblivious and anonymous protocols.

5.1 Protocol Descriptions and Security Definitions for Oblivious Databases

Our oblivious database protocols combine the scheme of Section 3.2 with a multi-receiver OT protocol. Each transaction is conducted between one of a collection of users and a single database server $\mathcal{D}$, who takes the place of the credential provider in our protocols. We describe the protocol specifications below.

1. $\mathsf{Setup}(\mathcal{U}(1^k), \mathcal{D}(1^k, \Pi_1, \ldots, \Pi_n, M_1, \ldots, M_N))$**:** The database server $\mathcal{D}$ generates parameters *params* for the scheme. As in the basic credential scheme, it generates a cryptographic representation $\Pi_i^{\mathcal{C}}$ for each policy graph, and publishes those values via an authenticated channel. In addition, $\mathcal{D}$ initializes its database of messages according to the OT protocol in use. Each user $\mathcal{U}$ generates a keypair and requests that it be certified by a trusted CA.
2. $\mathsf{ObtainCred}(\mathcal{U}(pk_{\mathcal{D}}, sk_{\mathcal{U}}, \Pi_i^{\mathcal{C}}), \mathcal{D}(pk_{\mathcal{U}}, sk_{\mathcal{D}}, \Pi_i^{\mathcal{C}}, S))$**:** $\mathcal{U}$ registers with the system and receives a credential Cred which binds her to a policy graph Π_i and starting state S.
3. $\mathsf{AccessAndUpdate}(\mathcal{U}(pk_{\mathcal{D}}, sk_{\mathcal{U}}, \mathsf{Cred}, j), \mathcal{D}(sk_{\mathcal{D}}, E))$**:** $\mathcal{U}$ requests an item at index j in the database from state S by selecting a tag $(\mathsf{pid}, S \to T, j)$ from the policy graph. The user then updates her credential Cred, in such a way that $\mathcal{D}$ does not learn her identity, or her current state. Simultaneously, $\mathcal{U}$ obtains a message from the database at index j. At the end of a successful protocol, $\mathcal{U}$ updates the state information in Cred, and $\mathcal{D}$ updates a local datastore E.

Security. We give both informal and formal security definitions.

Database Security: No (possibly colluding) subset of corrupted users can obtain any collection of items that is not specifically permitted by the users' policies.

User Security: A malicious database controlling some collection of corrupted users cannot learn any information about a user's identity or her state in the policy graph, beyond what is available through auxiliary information from the environment.

Definition 5.1 (Security for Oblivious Databases with Access Controls)**.** Security is defined according to the following experiments, which are based on those of [19]. Although we do not explicitly specify auxiliary input to the parties, this information can be provided in order to achieve sequential composition.

Real experiment. The real-world experiment $\mathbf{Real}_{\hat{\mathcal{D}},\hat{\mathcal{U}}_1,\ldots,\hat{\mathcal{U}}_\eta}(\eta, N, k, \Pi_1, \ldots, \Pi_\eta, M_1, \ldots, M_N, \Sigma)$ is modeled as k rounds of communication between a possibly cheating database $\hat{\mathcal{D}}$ and a collection of η possibly cheating users $\{\hat{\mathcal{U}}_1, \ldots, \hat{\mathcal{U}}_\eta\}$. $\hat{\mathcal{D}}$ is given the policy graph for each user $\Pi_1, \ldots, \Pi_\eta$, a message database $M_1, \ldots, M_N$ and the users are given an adaptive strategy Σ that, on input of the user's identity and current graph state, outputs the next action to be taken by the user.

At the beginning of the experiment, the database and users conduct the Setup and $\mathsf{ObtainCred}$ protocols. At the end of this step, $\hat{\mathcal{D}}$ outputs an initial state D_1, and each user $\hat{\mathcal{U}}_i$ outputs state $U_{1,i}$. For each subsequent round $j \in [2, k]$, each user may interact with $\hat{\mathcal{D}}$ to request an item $\alpha \in [1, N+1]$ as required by the strategy Σ. Following each round, $\hat{\mathcal{D}}$ outputs D_j, and the users output $U_{1,j}, \ldots, U_{\eta,j}$, respectively. At the end of the k^{th} round the output of the experiment is $(D_k, U_{1,k}, \ldots, U_{j,k})$.

We define the *honest* database $\mathcal{D}$ as one that honestly runs its portion of Setup and $\mathsf{ObtainCred}$ in the first round, honestly runs its side of the $\mathsf{AccessAndUpdate}$ protocol when requested by a user at round $j > 1$, and outputs $D_k = params$. Similarly, an honest user $\mathcal{U}_i$ runs the Setup and $\mathsf{ObtainCred}$ protocol honestly in the first round, executes the user's side of the $\mathsf{AccessAndUpdate}$ protocol in subsequent rounds, and eventually outputs the credential values and all database items received.

Ideal experiment. In experiment $\mathbf{Ideal}_{\hat{\mathcal{D}}',\hat{\mathcal{U}}_1',\ldots,\hat{\mathcal{U}}_\eta'}(\eta, N, k, \Pi_1, \ldots, \Pi\eta, M_1, \ldots, M_N, \Sigma)$ the possibly cheating database $\hat{\mathcal{D}}'$ first generates messages $(M_1^*, \ldots, M_{N+1}^*)$, with M_{N+1}^* representing the null item in the database. In the first round, $\hat{\mathcal{D}}'$ sends $(M_1^*, \ldots, M_{N+1}^*)$ and the policy graphs $\Pi_1, \ldots, \Pi_{eta}$, to the trusted party $\mathcal{T}$. In addition, all users $\hat{\mathcal{U}}_i'$ sends a join message of the form

(join, i) to $\mathcal{T}$, who forwards the message to $\hat{\mathcal{D}}'$ along with a bit $b_{\mathcal{T}}$ indicating the existence of the user's respective policy graph. $\hat{\mathcal{D}}'$ replies to $\mathcal{T}$ with a bit $b_{\hat{\mathcal{D}}'} \in \{0, 1\}$ indicating the success of the protocol, and the user's initial state S_i. If the protocol fails at any time, $S_i = \bot$. Finally, $\mathcal{T}$ sends $(b_{\hat{\mathcal{D}}'} \wedge b_{\mathcal{T}}, S_i)$ to $\hat{\mathcal{U}}_i'$.

In each round $j \in [2, k]$, every user $\hat{\mathcal{U}}_i'$ selects a message index $\alpha \in [1, N+1]$ and sends $(\mathsf{access}, i, S_i, T_i, \alpha, t_{id})$ to $\mathcal{T}$ according to the strategy Σ , where t_{id} represents a transaction id for restarting the protocol in case of a failure. $\mathcal{T}$ checks the user's policy graph Π_i to determine if $\hat{\mathcal{U}}_i'$ is in state S_i, and that the transition to T_i is permits access to the message at index α. If the transition and message selection are allowed by the user's policy or if the tuple has previously been seen, $\mathcal{T}$ sets $b_{\mathcal{T}} = 1$. Otherwise, $b_{\mathcal{T}} = 0$. $\mathcal{T}$ then sends $(\mathsf{access}, t_{id}, b_{\mathcal{T}})$ to $\hat{\mathcal{D}}'$, who returns a bit $b_{\hat{\mathcal{D}}'}$ determining whether the transaction should succeed. If $b_{\hat{\mathcal{D}}'} \wedge b_{\mathcal{T}} = 1$, then $\mathcal{T}$ returns M_α^* and 1 to $\hat{\mathcal{U}}_i'$. Otherwise it returns $\bot$ and 0. Following each round, $\hat{\mathcal{D}}'$ outputs D_j, and the users output $U_{1,j}, \ldots, U_{\eta,j}$, respectively. At the end of the k^{th} round the output of the experiment is $(D_k, U_{1,k}, \ldots, U_{\eta,k})$.

We will define the *honest* database $\mathcal{D}'$ as one that sends $M_1, \ldots, M_{N+1}$, with $M_{N+1} = \bot$, in the first round, returns $b_{\mathcal{D}'} = 1$ in all subsequent rounds, and outputs $D_k = \epsilon$. Similarly, an honest user $\mathcal{U}_i'$ runs the Setup and $\mathsf{ObtainCred}$ protocol honestly in the first round, makes queries and transitions according to the strategy Σ in subsequent rounds, and eventually outputs all received database items and credential values as its output.

Let $\ell(\cdot), c(\cdot), d(\cdot)$ be polynomials. We now define database and user security in terms of the experiments above.

Database Security. An oblivious transfer scheme with access controls is database-secure if for every collection of possibly cheating real-world p.p.t. users $\hat{\mathcal{U}}_1, \ldots, \hat{\mathcal{U}}_\eta$ there exists a collection of p.p.t. ideal-world users $\hat{\mathcal{U}}_1', \ldots, \hat{\mathcal{U}}_\eta'$ such that $\forall N = \ell(\kappa)$, $\eta = d(\kappa)$, $k \in c(\kappa)$, Σ, and every p.p.t. distinguisher:

$$\begin{aligned}&\mathbf{Real}_{\mathcal{D}, \hat{\mathcal{U}}_1, \ldots, \hat{\mathcal{U}}_\eta}(\eta, N, k, \Pi_1, \ldots, \Pi_\eta, M_1, \ldots, M_N, \Sigma) \overset{c}{\approx} \\ &\quad \mathbf{Ideal}_{\mathcal{D}', \hat{\mathcal{U}}_1', \ldots, \hat{\mathcal{U}}_\eta'}(\eta, N, k, \Pi_1, \ldots, \Pi_\eta, M_1, \ldots, M_N, \Sigma)\end{aligned}$$

User Security. An oblivious transfer scheme with access controls provides user security if for every real-world possibly cheating p.p.t. database $\hat{\mathcal{D}}$ and collection of possibly cheating users $\hat{\mathcal{U}}_1, \ldots, \hat{\mathcal{U}}_\eta$, there exists a p.p.t. ideal-world sender $\hat{\mathcal{D}}'$ and ideal users $\hat{\mathcal{U}}_1', \ldots, \hat{\mathcal{U}}_\eta'$ such that $\forall N = \ell(\kappa)$, $\eta = d(\kappa)$, $k \in c(\kappa)$, Σ, and every p.p.t. distinguisher:

$$\begin{aligned}&\mathbf{Real}_{\hat{\mathcal{D}}, \hat{\mathcal{U}}_1, \ldots, \hat{\mathcal{U}}_\eta}(\eta, N, k, \Pi_1, \ldots, \Pi_\eta, M_1, \ldots, M_N, \Sigma) \overset{c}{\approx} \\ &\quad \mathbf{Ideal}_{\hat{\mathcal{D}}', \hat{\mathcal{U}}_1', \ldots, \hat{\mathcal{U}}_\eta'}(\eta, N, k, \Pi_1, \ldots, \Pi_\eta, M_1, \ldots, M_N, \Sigma)\end{aligned}$$

5.2 The Construction

To construct our protocols, we extend the basic credential scheme of Section 3.2 by linking it to the adaptive OT protocol of [19]. The database operator commits to a collection of N messages, along with a special *null* message at index $N+1$. It then distributes these commitments, along with the cryptographic representation of the policy graph (*e.g.,* via a website). Each user then registers with the database using the $\mathsf{ObtainCred}$ protocol, and agrees to be bound by a policy that will control her ability to access the database.

$\mathsf{Setup}(\mathcal{U}(1^k), \mathcal{D}(1^k, \Pi_1, \ldots, \Pi_n, M_1, \ldots, M_N))$**:** When the database operator $\mathcal{D}$ is initialized with a database of messages $M_1, \ldots, M_N$, it conducts the following steps:

1. $\mathcal{D}$ selects parameters for the OT scheme as $\gamma = (q, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow \mathsf{BMsetup}(1^\kappa)$, $h \overset{\$}{\leftarrow} \mathbb{G}$, $x \overset{\$}{\leftarrow} \mathbb{Z}_q$, and $H \leftarrow e(g, h)$. $\mathcal{D}$ generates two CL signing keypairs $(spk_\mathcal{D}, ssk_\mathcal{D})$ and $(gpk_\mathcal{D}, gsk_\mathcal{D})$, and $\mathcal{U}$ generates her keypair $(pk_\mathcal{U}, sk_\mathcal{U})$ as in the credential Setup protocol of Figure 1.
2. For $j = 1$ to $(N + 1)$, $\mathcal{D}$ computes a ciphertext $C_j = (A_j, B_j)$ as:
   (a) If $j \leq N$, then $A_j = g^{\frac{1}{x+j}}$ and $B_j = e(h, A_j) \cdot M_j$.
   (b) If $j = (N + 1)$, compute A_j as above and set $B_j = e(h, A_j)$.
3. For every graph Π_i to be enforced, $\mathcal{D}$ generates a cryptographic representation $\Pi_i^\mathcal{C}$ as follows:
   (a) $\mathcal{D}$ parses Π_i to obtain a unique policy identifier pid.
   (b) For each tag $t = (\mathsf{pid}, S, T, j)$ with $j \in [1, N + 1]$, $\mathcal{D}$ computes the signature $\sigma_{S \to T, j} \leftarrow \mathsf{CLSign}(gsk_\mathcal{P}, (\mathsf{pid}, S, T, j))$. Finally, $\mathcal{D}$ sets $\Pi_i^\mathcal{C} \leftarrow \langle \Pi_i, \forall t : \sigma_{S \to T, j} \rangle$.
4. $\mathcal{D}$ sets $pk_\mathcal{D} = (spk_\mathcal{D}, gpk_\mathcal{D}, \gamma, H, y = g^x, C_1, \ldots, C_{N+1})$ and $sk_\mathcal{D} = (ssk_\mathcal{D}, gsk_\mathcal{D}, h)$. $\mathcal{D}$ then publishes each $\Pi_i^\mathcal{C}$ and the OT parameters $pk_\mathcal{D}$ via an authenticated channel.

$\mathsf{ObtainCred}(\mathcal{U}(pk_\mathcal{D}, sk_\mathcal{U}, \Pi_i^\mathcal{C}), \mathcal{D}(pk_\mathcal{U}, sk_\mathcal{D}, \Pi_i^\mathcal{C}, S))$**:** When user $\mathcal{U}$ wishes to join the system, it negotiates with $\mathcal{D}$ to agree on a policy Π_i and initial state S, then:

1. $\mathcal{U}$ picks a random show nonce $N_s \in \mathbb{Z}_q$ and computes $A \leftarrow \mathrm{Commit}(sk_\mathcal{U}, N_s)$.
2. $\mathcal{U}$ conducts an interactive proof to convince $\mathcal{D}$ that A correlates to $pk_\mathcal{U}$, and $\mathcal{D}$ conducts an interactive proof of knowledge to convince $\mathcal{U}$ that $e(g, h) = H$.
3. $\mathcal{U}$ and $\mathcal{P}$ run the CL signing protocol on committed values so that $\mathcal{U}$ obtains the state signature $\sigma_\mathsf{state} \leftarrow \mathsf{CLSign}(ssk_\mathcal{P}, (sk_\mathcal{U}, N_s, \mathsf{pid}, S))$ with pid, S contributed by $\mathcal{P}$.
4. $\mathcal{U}$ stores the credential $\mathsf{Cred} = (\Pi_i^\mathcal{C}, S, \sigma_\mathsf{state}, N_s)$.

Figure 6: Setup and user registration algorithms for an access controlled oblivious database based on the Camenisch, Neven and shelat oblivious transfer protocol [19]. The database operator and users first run the Setup portion of the protocol. Each user subsequently registers with the database using ObtainCred.

To obtain items from the database, the user runs the AccessAndUpdate protocol, which proves (in zero knowledge) that its request is consistent with its policy. Provided the user does not violate her policy, the user is assured that the database operator learns nothing about its identity, or the nature of its request. Figures 6 and 7 describe the protocols in detail.

Theorem 5.2. *The scheme described above satisfies database and user security (as defined in Definition 5.1) under the q-PDDH, q-SDH, and Strong RSA assumptions.*

Proof sketch. Our proof refers substantially to the original proof of the underlying adaptive oblivious transfer protocol due to [19]. Due to space limitations we sketch the full proof and refer the reader to the appropriate sections of that work. We consider the user and database security separately. We begin by describing the operation of a p.p.t. simulator for the ideal-world instantiations of the protocol, and then argue the computational indistinguishability between $\mathbf{Real}_{\hat{\mathcal{D}}, \hat{\mathcal{U}}_1, \ldots, \hat{\mathcal{U}}_\eta}$ and $\mathbf{Ideal}_{\hat{\mathcal{D}}', \hat{\mathcal{U}}'_1, \ldots, \hat{\mathcal{U}}'_\eta}$ via a hybrid argument. We denote the advantage of D in distinguishing the output of **Game i** as:

$$Pr[\textbf{Game i}] = Pr[D(X) = 1 : X \overset{\$}{\leftarrow} \textbf{Game i}]$$

User Security: To prove user security, we first describe a simulator for the ideal-world database $\hat{\mathcal{D}}'$ with rewindable black box access to a real-world database $\hat{\mathcal{D}}$. Upon initialization, $\hat{\mathcal{D}}$ outputs the system parameters, the cryptographic representation of the policy graphs for each user $\Pi_1^\mathcal{C}, \ldots, \Pi_\eta^\mathcal{C}$,

$\mathsf{AccessAndUpdate}(\mathcal{U}(pk_{\mathcal{D}}, sk_{\mathcal{U}}, \mathsf{Cred}, j), \mathcal{D}(pk_{\mathcal{D}}, E))$**:** When $\mathcal{U}$ wishes to obtain the message indexed by $j \in [1, N+1]$, it first identifies a tag t in Π_i such that $t = (\mathsf{pid}, S \to T, j)$.

1. $\mathcal{U}$ parses $\mathsf{Cred} = (\Pi_i^{\mathcal{C}}, S, \sigma_{\mathsf{state}}, N_s)$, and parses $\Pi_i^{\mathcal{C}}$ to find $\sigma_{S\to T,i}$.
2. $\mathcal{U}$ selects $N_s' \xleftarrow{\$} \mathbb{Z}_q$ and computes $A \leftarrow \text{Commit}(sk_{\mathcal{U}}, N_s', \mathsf{pid}, T)$.
3. $\mathcal{U}$ then sends (N_s, A) to $\mathcal{D}$. $\mathcal{D}$ checks the database E for $(N_s, A' \neq A)$, and if it finds such an entry it aborts. Otherwise it adds (N_s, A) to E.
4. $\mathcal{U}$ parses $C_j = (A_j, B_j)$. It selects a random $v \leftarrow \mathbb{Z}_q$ and sets $V \leftarrow (A_j)^v$. It sends V to $\mathcal{D}$ and proves knowledge of $(j, v, sk_{\mathcal{U}}, \sigma_{S\to T,j}, \sigma_{\mathsf{state}}, \mathsf{pid}, S, T, N_s')$ such that the following conditions hold:
    (a) $e(V, y) = e(g, g)^v e(V, g)^{-j}$.
    (b) $A = \text{Commit}(sk_{\mathcal{U}}, N_s', \mathsf{pid}, T)$.
    (c) $\mathsf{CLVerify}(spk_{\mathcal{P}}, \sigma_{\mathsf{state}}, (sk_{\mathcal{U}}, N_s, \mathsf{pid}, S)) = 1$.
    (d) $\mathsf{CLVerify}(\mathcal{P}, \sigma_{S\to T,j}, (\mathsf{pid}, S, T, j)) = 1$.
5. If these proofs verify, $\mathcal{U}$ and $\mathcal{D}$ run the CL signing protocol on committed values such that $\mathcal{U}$ obtains $\sigma'_{\mathsf{state}} \leftarrow \mathsf{CLSign}(ssk_{\mathcal{D}}, A)$. $\mathcal{U}$ stores the updated credential $\mathsf{Cred}' = (\Pi_i^{\mathcal{C}}, T, \sigma'_{\mathsf{state}}, N_s')$.
6. Finally, $\mathcal{D}$ returns $U = e(V, h)$ and interactively proves that U is correctly formed (see [19]). $\mathcal{U}$ computes the message $M_j = B_j / U^{1/v}$.

Figure 7: Database access protocol for an access-controlled oblivious database based on the Camenisch, Neven and shelat adaptive oblivious transfer protocol [19].

and commitments to the messages $C_1, \ldots, C_{N+1}$. The simulator $\hat{\mathcal{D}}'$ checks that the parameters, signatures, and commitments are valid. If any check fails, the simulator aborts. Otherwise, $\hat{\mathcal{D}}'$ generates a user keypair for a randomly chosen user and performs the $\mathsf{ObtainCred}$ protocol with $\hat{\mathcal{D}}$. $\hat{\mathcal{D}}'$ rewinds $\hat{\mathcal{D}}$ and uses the extractor for the zero-knowledge protocol to extract the value h from the proof that $e(g, h) = H$. Using h, $\hat{\mathcal{D}}'$ decrypts the values $C_1, \ldots, C_{N+1}$ to obtain the plaintext messages $M_1^*, \ldots, M_{N+1}^*$, where M_{N+1}^* is the null message. Finally, $\hat{\mathcal{D}}'$ sends the plaintext policy graphs $\Pi_1, \ldots, \Pi_\eta$ and messages $M_1^*, \ldots, M_{N+1}^*$ to the trusted part $\mathcal{T}$.

During round 1 of the simulation, $\hat{\mathcal{D}}'$ may receive a *join* message of the form $(join, i, b_{\mathcal{T}})$ from each of the η users. When $\hat{\mathcal{D}}'$ receives such a message, it generates a user keypair according to the procedure outlined in Setup for the user $\mathcal{U}_i$. $\hat{\mathcal{D}}'$ then runs the $\mathsf{ObtainCred}$ protocol with $\hat{\mathcal{D}}$ using $\mathcal{U}_i$'s policy graph and locally generated keypair. During the protocol, $\hat{\mathcal{D}}'$ rewinds $\hat{\mathcal{D}}$ and uses the extractor to reveal the value h from the zero-knowledge proof. If at any point the protocol fails or if this value is different than the value received during initialization, $\hat{\mathcal{D}}'$ aborts and sends $b_{\hat{\mathcal{P}}'} = 0$ and $S_i = \perp$ to $\mathcal{T}$. If the protocol succeeds, $\hat{\mathcal{D}}'$ checks the received state signature and sends the user's state S_i and $b_{\hat{\mathcal{P}}'} = 1$ to $\mathcal{T}$. $\hat{\mathcal{D}}'$ also stores the user's information, including the keypair that was generated, her policy graph, her start state, and the state signature returned after completion of the $\mathsf{ObtainCred}$ protocol. At the end of round 1, $\hat{\mathcal{P}}'$ will have a local copy of the current credentials for every user under the keys that were generated when the *join* message was received.

When $\hat{\mathcal{D}}'$ receives an $(access, t_{id}, b_{\mathcal{T}})$ message in rounds $2, \ldots, k$, it first checks to see if the value t_{id} has been seen previously. If it has, then $\hat{\mathcal{D}}'$ chooses a user at random from among those it has already used in an $\mathsf{AccessAndUpdate}$ protocol, and reuses the same commitment and nonce that was previously used. Otherwise, $\hat{\mathcal{D}}'$ selects a user uniformly at random from among all credentials it has stored, and chooses a valid transition from that user's current state (as represented in the locally stored credential) to its next state, along with the associated message index. Such a transition is guaranteed to exist for all users due to our use of null transitions on terminal states. Next $\hat{\mathcal{D}}'$ runs the $\mathsf{AccessAndUpdate}$ protocol with $\hat{\mathcal{D}}$, acting as the selected user with either a reused commitment and nonce pair, or a valid transition and associated message index. If the protocol fails at any point, $\hat{\mathcal{D}}'$ sends $b_{\hat{\mathcal{D}}'}$ to $\mathcal{T}$. If the protocol succeeds, $\hat{\mathcal{D}}'$ checks the state signature and received message for validity and returns $b_{\hat{\mathcal{D}}'} = 1$. In addition, $\hat{\mathcal{D}}'$ also stores the received signature as the user's updated credential.

Now, we examine the statistical difference between successive distributions. In **Game 0** = $\mathbf{Real}_{\hat{\mathcal{D}},\mathcal{U}_1,\ldots,\mathcal{U}_\eta}$, the cheating real-world database $\hat{\mathcal{D}}$ interacts with honest users $\mathcal{U}_1, \ldots, \mathcal{U}_\eta$.

Game 1 is exactly that of **Game 0** with the exception that we examine the soundness of the zero-knowledge proof of knowledge of the value h by the database in ObtainCred and during the initialization phase. To do so, we show that an extractor fails to extract the value h with probability equal to the (negligible) knowledge error of the zero-knowledge protocol. We define this probability as $\nu_1(\kappa)$.[6] Since this procedure needs to be run at initialization to recover the messages to send to $\mathcal{T}$ and once for each user's *join* request to ensure the value has not changed, we have:

$$Pr[\textbf{Game 1}] - Pr[\textbf{Game 0}] \leq \nu_1(\kappa)$$

Game 2 operates exactly as **Game 1**, except we replace the commitments used in the real-world protocol with our simulated commitments of randomly chosen user credentials maintained by $\hat{\mathcal{D}}'$. By the hiding property of the commitment scheme, these simulated commitments will be indistinguishable from real ones.[7] For some negligible function $\nu_2(\kappa)$, we have $Pr[\textbf{Game 2}] - Pr[\textbf{Game 1}] \leq \nu_2(\kappa)$.

In **Game 3** we operate as in **Game 2**, but replace the zero-knowledge proofs of the credential values provided by honest users with those of the credentials maintained by $\hat{\mathcal{D}}'$ in our simulator. Again, the crux of this argument lies in showing that the witnesses provided during the zero-knowledge proof protocols for our simulated values is identically distributed to those of the honest users. By the witness-indistinguishability property of zero-knowledge proofs of knowledge, we are guaranteed that $Pr[\textbf{Game 3}] - Pr[\textbf{Game 2}] < \nu_4(\kappa)$ for negligible $\nu_4(\kappa)$.[8]

We define **Game 4** to be identical to **Game 3**, but now we replace the state signatures produced by the blind signature scheme with an honest user with the signatures obtained by our simulator on the simulated user credentials. As with the previous components, the signatures produced by the blind signature must be uniformly distributed in the underlying group. However, we must also require that the interaction in the blind signatures scheme itself does not leak information about the contents of the message being signed. To do so, we can instantiate our signature scheme with the RSA-based [15], bilinear [16], or [5] blind signature schemes, and apply the transformations described by [32] to make them selective-failure blind signatures. These signatures are secure under the Strong RSA, LRSW, and Strong Diffie-Hellman assumptions, respectively. Therefore, we have $Pr[\textbf{Game 4}] - Pr[\textbf{Game 3}] < \nu_5(\kappa)$.

Finally, we define **Game 5** to be the interaction of our simulated ideal-world cheating server $\hat{\mathcal{D}}'$ with the trusted party $\mathcal{T}$. This interaction incorporates all of the changes outlined above, and we can bound D's advantage in distinguishing **Game 0** from **Game 5** by summing the statistical difference:

$$\begin{aligned} Pr[D(X) = 1 : X \xleftarrow{\$} \mathbf{Ideal}_{\hat{\mathcal{D}}',\mathcal{U}'_1,\ldots,\mathcal{U}'_\eta}] - Pr[D(X) = 1 : X \xleftarrow{\$} \mathbf{Real}_{\hat{\mathcal{D}},\mathcal{U}_1,\ldots,\mathcal{U}_\eta}] \\ \leq \nu_1(\kappa) + \nu_2(\kappa) + \nu_3(\kappa) + \nu_4(\kappa) + \nu_5(\kappa) \end{aligned}$$

Database Security: To prove database security, we first describe simulators for the ideal-world users $\hat{\mathcal{U}}'_1, \ldots, \hat{\mathcal{U}}'_\eta$ with rewindable black box access to the real-world users $\hat{\mathcal{U}}_1, \ldots, \hat{\mathcal{U}}_\eta$ using strategy

[6] When the zero-knowledge proof is implemented using the Schnorr technique in bilinear groups, the probability of failure is $1/q$, with q being the order of the group.

[7] When using [46] commitments, the commitment values are perfectly hiding.

[8] Many perfect zero-knowledge proof protocols exist, such as [26], which imply the witness indistinguishability property with no computational complexity assumptions.

Σ. For clarity, we refer to the simulator as $\hat{\mathcal{U}}_i'$ with access to $\hat{\mathcal{U}}_i$. Upon initialization, $\hat{\mathcal{U}}_i'$ generates the database keypairs according to the Setup protocol and signs each of the policy graphs to create their cryptographic representations $\Pi_1^{\mathcal{C}}, \ldots, \Pi_\eta^{\mathcal{C}}$. Additionally, $\hat{\mathcal{U}}_i'$ generates dummy commitment values $C_j = (A_j, B_j)$ by setting $A_j = g^{\frac{1}{x+j}}$ and B_j to be a random group element from $\mathbb{G}_T$ for all $j \in [1, N]$. The commitment value C_{N+1} for the null message is created as defined in AccessAndUpdate. $\hat{\mathcal{U}}_i'$ sends the parameters, policy graphs, and commitments to $\hat{\mathcal{U}}$.

In round 1, $\hat{\mathcal{U}}_i$ will initiate the ObtainCred protocol with $\hat{\mathcal{U}}_i'$. When the protocol is initiated, $\hat{\mathcal{U}}_i'$ will send a $(join, i)$ message to the trusted party $\mathcal{T}$. If $\mathcal{T}$ responds with a bit $b = 1$ and state S_i, $\hat{\mathcal{U}}_i'$ runs the protocol as usual and embeds the returned state S_i into the user's state signature. Otherwise, $\hat{\mathcal{U}}_i'$ aborts the protocol.

When $\hat{\mathcal{U}}_i$ initiates the AccessAndUpdate protocol in rounds $2, \ldots, k$, $\hat{\mathcal{U}}_i'$ runs the protocol as usual by checking for a reused update nonce, and verifying the user's proof of knowledge on her credential state. If the nonce was previously used and the commitment provided remains the same, $\hat{\mathcal{U}}_i'$ sets t_{id} to be the same as that which was used in the transaction with the repeated nonce, commitment pair. If the nonce is new, t_{id} is chosen randomly by $\hat{\mathcal{U}}_i'$. In addition, $\hat{\mathcal{U}}_i'$ rewinds $\hat{\mathcal{U}}_i$ and uses the extractor for the proof of knowledge to reveal the user's identity i, current state S_i, intended next state T_i, blinding factor v, and message index choice α. Then, $\hat{\mathcal{U}}_i'$ sends the message $(update, i, S_i, T_i, \alpha, t_{id})$ to $\mathcal{T}$. If $\mathcal{T}$ returns $b = 1$, $\hat{\mathcal{U}}_i'$ sends $U = B_\alpha^v / M_\alpha$ and the new state signature for state T_i to $\hat{\mathcal{U}}_i$. If $\hat{\mathcal{U}}_i'$ receives $b = 0$, it aborts the AccessAndUpdate protocol.

We now investigate the differences between successive distributions. We define **Game 0** = $\mathbf{Real}_{\mathcal{D},\hat{\mathcal{U}}_1,\ldots,\hat{\mathcal{U}}_\eta}$ as the distribution where the honest real-world database $\mathcal{D}$ interacts with the potentially cheating users $\hat{\mathcal{U}}_1, \ldots, \hat{\mathcal{U}}_\eta$. By the notation described above, we have

$$Pr[\mathbf{Game\ 0}] = Pr[D(X) = 1 : X \xleftarrow{\$} \mathbf{Real}_{\mathcal{D},\hat{\mathcal{U}}_1,\ldots,\hat{\mathcal{U}}_\eta}]$$

Game 1 considers the same distribution as **Game 0**, but rather than encrypting the true message database $(M_1, \ldots, M_N)$ (and dummy element with value "1") we set $B_1, \ldots, B_{N+1}$ to be random elements drawn uniformly from $\mathbb{G}_T$. Under the $(N+1)$-PDDH assumption, D's advantage in distinguishing the values $B_i, \ldots, B_{N+1}$ from random elements of $\mathbb{G}_T$ is at most negligible, this bounds D's advantage in distinguishing this game from the previous game to a negligible value that we define as $\nu_1(\kappa)$. We refer the reader to the proof of Claim (2) from [19] for a full proof of this proposition. As such, we have $Pr[\mathbf{Game\ 1}] - Pr[\mathbf{Game\ 0}] < \nu_1(\kappa)$.

Game 2 is defined exactly as **Game 1**, except that we consider the ability of our extractor to correctly reveal the values in the zero-knowledge proofs in the AccessAndUpdate protocol. If the extractor cannot extract the values, the simulator would be unable to pass the correct state information to $\mathcal{T}$ for further processing. The probability of failure for our extractor is bounded by the knowledge error of the proof of knowledge, which we define as $\nu_2(\kappa)$.[9] Since we must run the extractor once for every message received from each user during rounds $j = 2, \ldots, k$, the statistical difference is bounded by:

$$Pr[\mathbf{Game\ 1}] - Pr[\mathbf{Game\ 0}] \leq k\eta\nu_2(\kappa)$$

In **Game 3** we operate as in **Game 2**, but consider the ability of the user to alter the value of the commitment between the proof of knowledge discussed in **Game 2** and the blind signature protocol. If, for example, the user were able to prove knowledge of the correct state and then alter the commitment to receive a blind signature on a different state, the future rounds of the

[9] For the Schnorr technique in a bilinear group of prime order q, this error is $1/q$.

protocol would produce errors when validating the user's state with $\mathcal{T}$. The binding property of the commitment scheme prevents such tampering except with negligible probability $\nu_3(\kappa)$.[10] Thus, $Pr[\textbf{Game 3}] - Pr[\textbf{Game 2}] \leq k\eta\nu_3(\kappa)$.

We define **Game 4** to be identical to **Game 3**, though we now consider the ability of the user to forge a valid blind signature on her state information. A user who succeeds at this goal would produce a new signature not produced by the database, thus violating the existential unforgeability property of the blind signature scheme used. For the RSA-based [15], bilinear [16], and [5] blind signature schemes, the probability that an adversary forges is at most negligible under the Strong RSA, LRSW, and Strong Diffie-Hellman assumptions, respectively. We refer to this probability by $\nu_4(\kappa)$, and thus, $Pr[\textbf{Game 4}] - Pr[\textbf{Game 3}] \leq k\eta\nu_4(\kappa)$.

We define **Game 5** to be the same as **Game 4**, except we now consider the indistinguishability of our simulated proof of the structure of U from the real proof of the correctly formed value. If $Pr[\textbf{Game 5}] - Pr[\textbf{Game 4}]$ is non-negligible, this would imply the existence of an algorithm that distinguishes the real from simulated zero knowledge proofs, which contradicts the zero knowledge property of the proof of knowledge. We define $\nu_5(\kappa)$ as a negligible value, and thus $Pr[\textbf{Game 5}] = Pr[\textbf{Game 4}] \leq k\eta\nu_5(\kappa)$.

Finally, we define **Game 6** to be the interaction of our simulated ideal-world cheating users $\hat{\mathcal{U}}'_1, \ldots, \hat{\mathcal{U}}'_\eta$ with the trusted party $\mathcal{T}$. This interaction incorporates all of the changes outlined above, and we bound D's advantage in distinguishing **Game 0** from **Game 6** by summing the statistical difference:

$$Pr[D(X) = 1 : X \overset{\$}{\leftarrow} \mathbf{Ideal}_{\hat{\mathcal{P}}', \mathcal{U}'_1, \ldots, \mathcal{U}'_\eta}] - Pr[D(X) = 1 : X \overset{\$}{\leftarrow} \mathbf{Real}_{\hat{\mathcal{P}}, \mathcal{U}_1, \ldots, \mathcal{U}_\eta}] \leq \\ \nu_1(\kappa) + k\eta(\nu_2(\kappa) + \nu_3(\kappa) + \nu_4(\kappa) + \nu_5(\kappa))$$

□

6 Conclusion

In this paper, we presented a flexible and efficient system that adds access controls and auditing onto oblivious transfer protocols. The flexibility of our approach makes it relatively straightforward to apply to a diverse set of anonymous and oblivious protocols. Our techniques can be easily extended to auditing and controlling blind signature schemes [15, 16, 5, 49], identity-based key extraction [38] and keyword search protocols [45]. This work takes a significant step forward in the important balance of user privacy and provider control.

Acknowledgements

The work of Scott Coull was supported in part by the U.S. Department of Homeland Security Science & Technology Directorate under Contract No. FA8750-08-2-0147. Matthew Green and Susan Hohenberger gratefully acknowledge the support of NSF grant CNS-0716142 and a Microsoft New Faculty Fellowship.

[10] This property holds under the discrete logarithm assumption for [46] commitments and the factoring assumption for [33] commitments.

References

[1] William Aiello, Yuval Ishai, and Omer Reingold. Priced oblivious transfer: How to sell digital goods. In *EUROCRYPT*, volume 2045, pages 119–135, 2001.

[2] D. Elliot Bell and Leonard J. La Padula. Secure Computer System: Unified Exposition and Multics Interpretation. *Communications of the ACM*, 1:271–280, 1988.

[3] K. J. Biba. Integrity Considerations for Secure Computer Systems. Technical Report ESD-TR-76-372, MITRE Corporation, April 1977.

[4] I. F. Blake and V. Kolesnikov. Strong Conditional Oblivious Transfer and Computing on Intervals. *ASIACRYPT '04*, 3329 of LNCS:515–529, 2004.

[5] Dan Boneh and Xavier Boyen. Short signatures without random oracles. In *EUROCRYPT '04*, volume 3027, pages 382–400, 2004.

[6] Fabrice Boudot. Efficient proofs that a committed number lies in an interval. In *EUROCRYPT '00*, volume 1807 of LNCS, pages 431–444, 2000.

[7] Stefan Brands. Rapid demonstration of linear relations connected by boolean operators. In *EUROCRYPT '97*, volume 1233 of LNCS, pages 318–333, 1997.

[8] David F. C. Brewer and Michael J. Nash. The Chinese Wall Security Policy. In *IEEE Symposium on Security and Privacy*, pages 206–214, 1989.

[9] Jan Camenisch and Ivan Damgård. Verifiable Encryption, Group Encryption, and Their Applications to Separable Group Signatures and Signature Sharing Schemes. In *ASIACRYPT '00*, volume 1976, pages 331–345, 2000.

[10] Jan Camenisch, Maria Dubovitskaya, and Gregory Neven. Oblivious Transfer with Access Controls. In *ACM Conference on Computer and Communications Security*, pages 131–140, 2009.

[11] Jan Camenisch, Susan Hohenberger, Markulf Kohlweiss, Anna Lysyanskaya, and Mira Meyerovich. How to win the clonewars: Efficient periodic n-times anonymous authentication. In *ACM CCS '06*, pages 201–210, 2006.

[12] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Compact e-cash. In *EUROCRYPT '05*, volume 3494 of LNCS, pages 302–321, 2005.

[13] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Balancing accountability and privacy using e-cash. In *SCN '06*, volume 4116, pages 141–155, 2006.

[14] Jan Camenisch and Anna Lysyanskaya. Efficient non-transferable anonymous multi-show credential system with optional anonymity revocation. In *EUROCRYPT '01*, volume 2045 of LNCS, pages 93–118. Springer, 2001.

[15] Jan Camenisch and Anna Lysyanskaya. A signature scheme with efficient protocols. In *Security in Communication Networks '02*, volume 2576, pages 268–289, 2002.

[16] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *CRYPTO '04*, volume 3152, pages 56–72, 2004.

[17] Jan Camenisch and Markus Michels. Proving in zero-knowledge that a number n is the product of two safe primes. In *EUROCRYPT '99*, volume 1592 of LNCS, pages 107–122, 1999.

[18] Jan Camenisch and Markus Michels. Separability and efficiency for generic group signature schemes. In *CRYPTO '99*, volume 1666, pages 413–430, 1999.

[19] Jan Camenisch, Gregory Neven, and Abhi Shelat. Simulatable adaptive oblivious transfer. In *EUROCRYPT*, volume 4515, pages 573–590, 2007.

[20] Jan Leonhard Camenisch. *Group Signature Schemes and Payment Systems Based on the Discrete Logarithm Problem.* PhD thesis, ETH Zürich, 1998.

[21] Agnes Chan, Yair Frankel, and Yiannis Tsiounis. Easy come – easy go divisible cash. In *EUROCRYPT '98*, volume 1403 of LNCS, pages 561–575, 1998.

[22] David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, October 1985.

[23] David Chaum and Torben Pryds Pedersen. Wallet databases with observers. In *CRYPTO*, volume 740 of LNCS, pages 89–105, 1992.

[24] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, 1998.

[25] D. D. Clark and D. R. Wilson. A Comparison of Commercial and Military Computer Security Policies. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 27–29, May 1987.

[26] R. Cramer, I. Damgard, and P. MacKenzie. Efficient Zero-Knowledge Proofs of Knowledge Without Intractability Assumptions. In *Public Key Cryptography*, volume 1751, pages 354–372, 2000.

[27] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO*, volume 839, pages 174–187, 1994.

[28] Giovanni Di Crescenzo, Rafail Ostrovsky, and S. Rajagopolan. Conditional oblivious transfer and time released encryption. In *EUROCRYPT '99*, volume 1592, pages 74–89, 1999.

[29] Ivan Damgård and Eiichiro Fujisaki. An integer commitment scheme based on groups with hidden order. In *ASIACRYPT*, volume 2501, pages 125–142, 2002.

[30] Department of Defense. Trusted Computer System Evaluation Criteria. Technical Report DoD 5200.28-STD, Department of Defense, December 1985.

[31] Yevgeniy Dodis and Aleksandr Yampolskiy. A Verifiable Random Function with Short Proofs and Keys. In *PKC '05*, volume 3386 of LNCS, pages 416–431, 2005.

[32] Marc Fischlin and Dominique Schröder. Security of Blind Signatures Under Aborts. In *Public Key Cryptography*, volume 5443, pages 297–316, 2009.

[33] Eiichiro Fujisaki and Tatsuaki Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In *CRYPTO*, volume 1294, pages 16–30, 1997.

[34] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to Construct Random Functions. *Journal of the ACM*, 33(4):792–807, 1986.

[35] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design. In *FOCS '86*, pages 174–187, 1986.

[36] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Computing*, 17(2), 1988.

[37] Google. Google Health. `https://www.google.com/health`, 2009.

[38] Matthew Green and Susan Hohenberger. Blind identity-based encryption and simulatable oblivious transfer. In *ASIACRYPT*, volume 4833, pages 265–282, 2007.

[39] Stanislaw Jarecki and Xiaomin Liu. Efficient Oblivious Pseudorandom Function with Applications to Adaptive OT and Secure Computation of Set Intersection. In *TCC*, volume 5444, pages 577–594, 2009.

[40] Butler W. Lampson. Dynamic Protection Structures. In *AFIPS Conference*, volume 35, pages 27–38, 1969.

[41] Anna Lysyanskaya. *Signature schemes and applications to cryptographic protocol design.* PhD thesis, MIT, Cambridge, Massachusetts, September 2002.

[42] Microsoft. Microsoft HealthVault. `http://www.healthvault.com/`, 2009.

[43] Moni Naor and Benny Pinkas. Oblivious transfer with adaptive queries. In *CRYPTO '99*, volume 1666 of LNCS, pages 573–590, 1999.

[44] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *FOCS '97*, pages 458–467, 1997.

[45] Wakaha Ogata and Kaoru Kurosawa. Oblivious keyword search. *Special issue on coding and cryptography J. of Complexity*, 20(2-3):356–371, 2004.

[46] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, volume 576, pages 129–140, 1992.

[47] Claus-Peter Schnorr. Efficient signature generation for smart cards. *J. of Cryptology*, 4(3):239–252, 1991.

[48] I. Teranishi, J. Furukawa, and K. Sako. k-Times Anonymous Authentication. In *ASIACRYPT 2004*, volume 3329, pages 308–322, 2004.

[49] Brent Waters. Efficient Identity-Based Encryption without random oracles. In *EUROCRYPT '05*, volume 3494 of LNCS, pages 114–127, 2005.